

UNISYS

A Series

Network Definition
Language II (NDLII)

**Programming Reference
Manual**

Release Mark 3.8.0

Priced Item

July 1989
Distribution Code SE
Printed in U S America
1169604.380

Product Information Announcement

☐ New Release ☐ Revision ☒ Update ☐ New Mail Code

Title**A Series Network Definition Language II (NDLII) Programming Reference Manual**

This Product Information Announcement announces the release of Update 5 to the *A Series Network Definition Language II (NDLII) Programming Reference Manual*, dated July 1989, relative to the Mark 3.8.0 System Software Release.

This manual documents the high-level programming and definition language used to describe a data communications network. It provides a brief overview and functional description of the data comm system and gives a complete description of the syntax and semantics of all language components and compiler options of NDLII. This manual is written for experienced data communications programmers.

Update 5 updates the ENUMERATION declaration to include the CODE_EXT_TYPE enumerated type, which is a data type of the predeclared variables RECEIVER.EXTEND and TRANSMITTER.EXTEND in adapter control. Recommendations regarding the use of the line STATION list attribute associated with the LINE definition are also included. Additional changes clarify the definition of a network information file II (NIFII) and update the data communications adapter (DCA) references to data communications host adapter (DCHA).

Changes to the text are indicated by vertical bars in the margins of the replacement pages.

Remove

iii through viii
xiii through xviii
1-1 through 1-2
1-5 through 1-6
1-9 through 1-10
2-9 through 2-12
3-83 through 3-88
4-17 through 4-18
4-39 through 4-40
B-1 through B-2
B-15 through B-20

Glossary-1 through 14
Bibliography-1 through 2
Index-1 through 18

Insert

iii through viii
xiii through xviii
1-1 through 1-2
1-5 through 1-6
1-9 through 1-10
2-9 through 2-12
3-83 through 3-88
4-17 through 4-18
4-39 through 4-40
B-1 through B-2
B-15 through B-20
B-21 through B-22

Glossary-1 through 14
Bibliography-1 through 2
Index-1 through 18

Retain this Product Information Announcement as a record of changes made to the basic publication.

To order additional copies of this document,

- United States customers call Unisys Direct at 1-800-228-9224.
- All other customers contact your Unisys Subsidiary Librarian.
- Unisys personnel use the Electronic Literature Ordering (ELO) system.

To receive the update package only, order 1169604-005. To receive the complete guide, order 1169604.380.

UNISYS

A Series

Network Definition
Language II (NDLII)

**Programming Reference
Manual**

Copyright © 1989 Unisys Corporation.

All rights reserved.

Unisys is a registered trademark of Unisys Corporation.

Release Mark 3.8.0

Priced Item

July 1989

Distribution Code SE

Printed in U S America

1169604.380

The names, places, and/or events used in this publication are not intended to correspond to any individual, group, or association existing, living, or otherwise. Any similarity or likeness of the names, places, and/or events with the names of any individual living or otherwise, or that of any group or association, is purely coincidental and unintentional.

NO WARRANTIES OF ANY NATURE ARE EXTENDED BY THE DOCUMENT. Any product and related material disclosed herein are only furnished pursuant and subject to the terms and conditions of a duly executed Program Product License or Agreement to purchase or lease equipment. The only warranties made by Unisys, if any, with respect to the products described in this document are set forth in such License or Agreement. Unisys cannot accept any financial or other responsibility that may be the result of your use of the information in this document or software material, including direct, indirect, special, or consequential damages.

You should be very careful to ensure that the use of this information and/or software material complies with the laws, rules, and regulations of the jurisdictions with respect to which it is used.

The information contained herein is subject to change without notice. Revisions may be issued to advise of such changes and/or additions.

Correspondence regarding this publication may be forwarded using the Product Information card at the back of the manual, or may be addressed directly to Unisys, Technical Publications, 460 Sierra Madre Villa, Pasadena, CA 91109.

Page Status

Page	Issue
iii through iv	-005
v	Original
vi through viii	-005
ix through xii	Original
xiii through xvii	-005
xviii	Blank
xix	Original
xx	Blank
xxi	Original
xxii	Blank
1-1	-005
1-2 through 1-5	Original
1-6	-005
1-7 through 1-8	Original
1-9	-005
1-10	Original
2-1 through 2-9	Original
2-10 through 2-12	-005
2-13 through 2-44	Original
3-1 through 3-82	Original
3-83 through 3-84	-005
3-85	Original
3-86 through 3-87	-005
3-88 through 3-107	Original
3-108	Blank
4-1 through 4-17	Original
4-18	-005
4-19 through 4-39	Original
4-40	-005
4-41 through 4-45	Original
4-46	Blank
A-1 through A-5	Original
A-6	Blank

continued

continued

Page	Issue
B-1	-005
B-2 through B-15	Original
B-16 through B-21	-005
B-22	Blank
C-1 through C-4	Original
D-1 through D-10	Original
E-1 through E-13	Original
E-14	Blank
F-1 through F-9	Original
F-10	Blank
Glossary-1 through 13	-005
Glossary-14	Blank
Bibliography-1	-005
Bibliography-2	Blank
Index-1 through 17	-005
Index-18	Blank

About This Manual

Purpose

Network Definition Language II (NDLII) is a high-level programming and definition language that is used to describe a data communications network. NDLII has been implemented for use on A Series and B 7900 systems that use network support processors (NSPs), line support processors (LSPs), and data communications data link processors (DCDLPs) within the input/output data communications (IODC) subsystem.

Scope

This manual is intended as a reference manual for NDLII. As such, its purpose is to present a complete description of the syntax and semantics of all language components and compiler options of NDLII, within a framework that is designed for quick access of information. The manual also contains an overview and a functional description of the NSP/LSP/DCDLP-based data comm subsystem in which NDLII programs run.

Audience

This manual is written for experienced data comm programmers who have a detailed knowledge of A Series systems data comm or general data comm concepts.

Prerequisites

To most effectively use this manual, data comm programmers should be familiar with the following concepts: terminal addresses, line speeds, vertical and horizontal parity, line polling/select procedures, and terminal hardware characteristics. A knowledge of message control system (MCS) concepts and of the DCALGOL programming language is also helpful.

How to Use This Manual

This manual can be read sequentially if you want to gain an overall picture of the NDLII program, or can be accessed randomly as a spot reference tool. When you access the manual randomly, be sure to make use of the table of contents and the index as guides to information.

The notation used in this manual to represent the syntax of NDLII is the railroad syntax diagram, which is frequently used in Unisys manuals. For those unfamiliar with this notation, a complete description is provided in Appendix F, "Understanding Railroad Diagrams."

The network information file II (NIFII) refers to the second generation of the network information file (NIF). NIFII is described in Section 1 of this manual.

SOURCENDLII, referenced in this manual, is an NDLII program supported by Unisys that provides an example of how an NDLII program can be written. Any comments or suggestions regarding SOURCENDLII should be submitted on a User Communication Form (UCF) with the Class specified as 2, the Type specified as 1, and the Product specified as *SOURCENDLII*.

In this guide notice that the spellings *adapter* and *adaptor* are both used. *Adapter* is used for Data Communications Host Adapters (DCHAs), and *adaptor* is used in other cases (such as line control adaptors). The latter spelling is affected by NDLII software conventions.

Organization

This manual is divided into the following sections and appendixes.

Section 1. NDLII Concepts and Structures

This section illustrates the basic structure of an NDLII program. It identifies and discusses the two modules that make up the NDLII source program. This section also describes the NDLII compiler and NDLII post compiler (NPC) and their functions, and introduces the NSP/LSP, DCDLP, enhanced data communications data link processor (EDCDLP), and data communications host adapter (DCHA) data comm subsystem.

Section 2. Common Language Components

This section describes the common language components, data types, general declarations, expressions, and statements that are used throughout this manual.

Section 3. Protocol Module

This section gives syntax and semantics for protocol module declarations, editor processes, line control processes, and adaptor control processes.

Section 4. Configuration Module

This section gives syntax and semantics for required definitions in the configuration module, which includes attributes for line, group, station, stationset, and terminal definitions.

Appendix A. System Messages

This appendix gives a list, with explanations, of NDLII run-time error messages.

Appendix B. NDLII Compiler and NDLII Post Compiler

This appendix describes the files used by the NDLII compiler and gives the syntax and semantics of the compiler control options for the NDLII compiler. It also describes the files used by the NPC for systems using EDCDLs or DCHAs, and describes how to run the NPC.

Appendix C. Reserved and Recognized Words

This appendix gives lists of reserved and recognized words for the protocol module and a list of reserved words for the configuration module.

Appendix D. NDLII Language Components

This appendix gives a list of NDLII compiler control options and lists declarations and statements that you can use in various parts of an NDLII program.

Appendix E. NDLIIANALYZER

This appendix gives the syntax and semantics required to initiate and operate the NDLIIANALYZER program. The appendix includes examples of the output from NDLIIANALYZER.

Appendix F. Understanding Railroad Diagrams

This appendix describes the notation used throughout the manual to represent the syntax of NDLII commands.

In addition, a glossary, a bibliography, and an index appear at the end of this manual.

Related Product Information

A Series Data Communications Protocols Installation and Implementation Guide (form 8600 0486)

This guide describes the purpose of protocols and procedures for installing protocols. It also provides reference material useful in interpreting dumps associated with data communications data link processors (DCDLs), enhanced data communications data link processors (EDCDLs), and data communications adapters (DCAs). This guide is written for system analysts and operations specialists who work with data communications.

A Series DCALGOL Programming Reference Manual (form 8600 0841)

This manual describes the Data Communications ALGOL (DCALGOL) language. This language is designed to support the implementation of message control systems (MCSs) and other resource monitoring and controlling programs that require access to special operating system interfaces. This manual is written for systems programmers.

About This Manual

A Series Interactive Datacomm Configurator (IDC) Operations Guide **(form 1169810)**

This guide explains how to use IDC, a menu-driven utility used to define and modify data communications networks. It provides information on configuring a data communications network using the IDC menu system and basic constructs, and provides reference information about the commands and attributes. This guide is written for individuals who have a basic knowledge of data communications concepts, but who might not know the physical characteristics of hardware devices in the network.

A Series Work Flow Language (WFL) Programming Reference Manual (form 8600 1047)

This manual presents the complete syntax and semantics of WFL. WFL is used to construct jobs that compile or run programs written in other languages and that perform library maintenance such as copying files. This manual is written for individuals who have some experience with programming in a block-structured language such as ALGOL and who know how to create and edit files using CANDE or the Editor.

Contents

About This Manual	v
 Section 1. NDLI Concepts and Structures	
NDLI Program Structure	1-2
Protocol Module Structure	1-3
Configuration Module Structure	1-4
NDLI Source Program Example	1-4
SOURCENDLI	1-5
NDLI Compiler	1-5
NDLI Post Compiler	1-6
The Data Comm Subsystem	1-7
Data Comm Subsystem Overview	1-7
Data Comm Subsystem Modules	1-7
Data Comm Subsystem Software Requirements	1-9
Data Communications Controller (DCC)	1-10
Message Control System (MCS)	1-10
 Section 2. Common Language Components	
Data Types	2-1
Language Elements	2-2
Constants	2-4
General Declarations	2-6
BOOLEAN Declaration	2-6
BYTE Declaration	2-7
DEFINE Declaration	2-8
ENUMERATION Declaration	2-10
Enumerated Variable Declaration	2-13
INTEGER Declaration	2-14
PROCEDURE Declaration	2-15
PROCESS Declaration	2-16
STRING Declaration	2-17
STRUCTURE Declaration	2-18
Variable Declaration List	2-19
Qualified Variables	2-20
Expressions	2-21
Boolean Expression	2-21
Byte Expression	2-25
Enumerated Expression	2-26
Integer Expression	2-27
Byte-to-Integer Promotion	2-28
LENGTH Function	2-28
Masking Functions	2-29
MAX Function	2-29

MIN Function	2-30
Rotate Function	2-30
Shift Function	2-30
Special Integer Functions	2-31
String Expression	2-32
Character Set	2-34
General Statements	2-35
ABORT Statement	2-35
Assignment Statement	2-36
Boolean Assignment	2-36
Byte Assignment	2-36
Enumerated Assignment	2-37
Integer Assignment	2-38
String Assignment	2-38
CALL Statement	2-38
CASE Statement	2-39
Compound Statement	2-40
DO Statement	2-41
IF Statement	2-42
SHORTEN Statement	2-43
SKIP Statement	2-43
WHILE Statement	2-44

Section 3. Protocol Module

Declarations in the Protocol Module	3-2
ABORTTYPE Declaration	3-3
APPLICATIONTYPE Declaration	3-3
CLASS Declaration	3-4
Asynchronous Mode	3-5
Bit Mode	3-6
Synchronous Mode	3-7
Line Class Attributes	3-9
ADDRESSMODE (BIT Only)	3-9
BITRATE (ASYNC and BIT Only)	3-10
CODE (All)	3-10
CONTROLMODE (BIT Only)	3-11
DIAGNOSTICRATE (SYNC Only)	3-11
NRZI (BIT Only)	3-11
PARITY (ASYNC and SYNC Only)	3-12
STOPBITS (ASYNC Only)	3-14
SYNCCONTROL (SYNC Only)	3-14
INCLUDE Declaration	3-15
SIGNAL and REPLY Declarations (Global)	3-16
TERMINALTYPE Declaration	3-18
TRANSLATETABLE Declaration	3-19
Editors	3-22
Editor Declarations	3-23
Predeclared Structure Variables in Editors	3-23
Editor Predeclared Variables	3-24
BUFFEROVERFLOW (Boolean, Read-Only)	3-24

DESTINATION (String, Read/Write)	3-24
SOURCE (String, Read-Only)	3-24
Special Editor Statements	3-24
Editor ALLOCATE Statement	3-24
Editor TRANSFER Statement	3-25
Editor TRANSLATE Statement	3-26
Special Editor Functions	3-26
DIV Function	3-27
DROP Function	3-27
HEAD Function	3-28
INTEGER Function	3-28
MOD Function	3-28
STRING Function	3-29
STRINGADD Function	3-29
STRINGSUBTRACT Function	3-30
SUBSTRING Function	3-30
TAIL Function	3-31
TAKE Function	3-31
Translate Functions	3-32
TRIM Function	3-32
Editor Process Fault Termination	3-33
Algorithms	3-34
ALGORITHM Declaration	3-34
DUPLICATE Structure Declaration	3-35
SIGNAL and REPLY Declarations	3-35
Line Control	3-36
LINE CONTROL Definition	3-36
Line Control Declarations	3-38
Predeclared Control Block Structure Variables	3-39
ABORTREASON (ABORTTYPE, Read-Only)	3-39
CATEGORY (CBCATEGORY, Read-Only)	3-39
DCE (Structure, Read-Only)	3-39
TEXT (Reference)	3-40
TEXTSIZE (Integer, Read-Only)	3-40
Predeclared GROUP Structure Variables	3-41
LINECOUNT (Integer, Read-Only)	3-41
MAXINPUT (Integer, Read-Only)	3-41
Predeclared LINE Structure Variables	3-42
ADAPTOREVENT (Event)	3-42
ADAPTORREADY (Boolean, Read-Only)	3-42
BUSY (Boolean, Read/Write)	3-42
CONNECTED (Boolean, Read-Only)	3-42
DISCONNECTION	
(DISCONNECTPOLICY, Read/Write)	3-42
READY (Boolean, Read-Only)	3-43
STATUSCHANGED (Boolean, Read/Write)	3-43
SYSTEM (Event)	3-44
Predeclared STATION Structure Variables	3-45
CONTROL (Byte, Read-Only)	3-45
ENABLED (Boolean, Read-Only)	3-45
INPUTACTION (INPUTPOLICY, Read-Only)	3-45
LINEWIDTH (Integer, Read-Only)	3-45

MAXINPUT (Integer, Read-Only)	3-45
MAXOUTPUT (Integer, Read-Only)	3-45
PAGECOUNT (Integer, Read-Only)	3-46
PAGESIZE (Integer, Read-Only)	3-46
QUEUED (Boolean, Read-Only)	3-46
READY (Boolean, Read-Only)	3-46
RECEIVEADDRESS (String, Read-Only)	3-46
REQUEST (Reference)	3-46
REQUESTLIST (List)	3-47
RESULT (Reference)	3-47
RESULTLIST (List)	3-48
SCREEN (Boolean, Read-Only)	3-48
TRANSMITADDRESS (String, Read-Only)	3-48
TYPE (TERMINALTYPE, Read-Only)	3-48
USAGECOUNT (Integer, Read-Only)	3-48
WRAPAROUND (Boolean, Read-Only)	3-48
Line Control Predeclared Variables	3-49
NOMEMORY (Boolean, Read-Only)	3-49
SYSTEMWAIT (Boolean, Read-Only)	3-49
TIMESTAMP (String[8], Read-Only)	3-49
Station Reference Variables	3-50
Special Line Control Declarations	3-51
CONTROL BLOCK Declaration	3-51
EVENT Declaration	3-52
INCLUDE Declarations in Line Control	3-53
INTERLOCK Declaration	3-53
PROCESS Declarations in Line Control	3-54
STRUCTURE Declarations in Line Control	3-54
Variable Declarations in Line Control	3-54
Special Line Control Statements	3-55
ALLOCATE Statement	3-56
CAUSE STATEMENT	3-56
CLEAR STATION Statement	3-57
COPYSTRING Statement	3-57
COPYSTRUCTURE Statement	3-58
DEALLOCATE Statement	3-58
DEQUEUE Statement	3-59
DISCARDRESULT Statement	3-59
ENQUEUE Statement	3-60
FINISHREQUEST Statement	3-61
INITIATE Statement	3-61
LOCK Statement	3-64
MOVETEXT Statement	3-65
NEWRESULT Statement	3-67
NEXTREQUEST Statement	3-67
NEXTSTATION Statement	3-68
NULLSTATION Statement	3-69
PROCESS Statement	3-69
PURGE Statement	3-70
REBLOCK Statement	3-71
SENDBLOCK Statement	3-72
TRACE Statement	3-72

UNLOCK Statement	3-73
WAIT Statement	3-73
Special Line Control Functions	3-74
HAPPENED Function	3-74
NULLREFERENCE Function	3-74
TIMEINTERVAL Function	3-75
WAIT Function	3-76
Line Control Fault Termination	3-77
Adaptor Control	3-78
ADAPTOR CONTROL Definition	3-79
Adaptor Control Declarations	3-80
Predeclared CB Structure Variable	3-81
Predeclared RECEIVER Structure Variables	3-81
BCS (Integer, Read/Write)	3-81
DLETECT (Boolean, Read-Only)	3-81
ECHOPLEX (Boolean, Read/Write)	3-81
ENABLED (Boolean, Read-Only)	3-81
ERROR (Structure, Read-Only)	3-82
EXTEND (CODE_EXT_TYPE, Read/Write) ..	3-83
FIRSTERROR (RTERROR, Read-Only)	3-83
IDLETECT (Boolean, Read-Only)	3-83
INHIBIT (Boolean, Read-Only)	3-83
LINECHAR (Byte, Read-Only)	3-84
RESIDUE (Integer, Read/Write)	3-84
SPECIAL (Boolean, Read-Only)	3-84
TRANSLATE (Boolean, Read/Write)	3-84
TRANSPARENT (Boolean, Read/Write) ...	3-84
Predeclared TRANSMITTER Structure Variables .	3-85
BCS (Integer, Read/Write)	3-85
DIAGNOSE (Boolean, Read/Write)	3-85
ENABLED (Boolean, Read-Only)	3-85
ERROR (Structure, Read-Only)	3-85
EXTEND (CODE_EXT_TYPE, Read/Write) ..	3-86
FIRSTERROR (RTERROR, Read-Only)	3-86
INHIBIT (Boolean, Read-Only)	3-87
LINECHAR (Byte, Read-Only)	3-87
RESIDUE (Integer, Read/Write)	3-87
SPECIAL (Boolean, Read/Write)	3-87
TRANSLATE (Boolean, Read/Write)	3-87
TRANSPARENT (Boolean, Read/Write) ...	3-87
Adaptor Control Predeclared Variables	3-88
BUFFEROVERFLOW (INPUT Process Only; Boolean, Read-Only)	3-88
ENDTEXT (OUTPUT Process Only; Boolean, Read-Only)	3-88
FETCHPARITY (Boolean, Read-Only)	3-88
NOCANCEL (Boolean, Read/Write)	3-88
Adaptor Control Process Declarations	3-88
Special Adaptor Control Statements	3-89
CANCEL Statement	3-89
DISABLE Statement	3-90
ENABLE Statement	3-91

FETCH Statement	3-92
INITIALIZE Statement	3-92
LOADSTRUCTURE Statement	3-93
RECEIVE Statement	3-94
STORE Statement	3-98
TRANSMIT Statement	3-99
Adaptor Control WAIT Statement	3-103
Special Adaptor Control Functions	3-104
BCSOF Function	3-104
Adaptor Control Translate Functions	3-105
WAITFORIDLE Function	3-106
Adaptor Control Fault Termination	3-107

Section 4. Configuration Module

ATTRIBUTE Definition	4-2
Default Definitions	4-4
DEFAULT LINE Definition	4-4
DEFAULT STATION Definition	4-4
DEFAULT TERMINAL Definition	4-5
FILE Definition	4-6
GROUP Definition	4-7
Group ALGORITHM Attribute	4-7
Group INITIALIZE Attribute	4-8
Group LINE ID Attribute	4-8
LINE Definition	4-9
Line DEFAULT Specification	4-9
Line ALGORITHM Attribute	4-10
Line CLASS Attribute	4-10
Line DISCONNECTACTION Attribute	4-11
Line DPRDELAY Attribute	4-11
Line FUNCTION Attribute	4-12
Line INITIALIZE Attribute	4-13
Line LOSSOFCARRIER Attribute	4-14
Line PHONE Attribute	4-15
Line READY Attribute	4-16
Line RECEIVEDDELAY Attribute	4-16
Line SETDIGITDELAY Attribute	4-17
Line STATION List	4-18
Line TIMEOUT Attribute	4-19
Line TRANSMITDELAY Attribute	4-19
STATION Definition	4-20
Station DEFAULT Specification	4-21
Station ADDRESS Attribute	4-21
Station ALGORITHM Attribute	4-22
Station APPLICATIONLIST Attribute	4-23
Station CONTROL Attribute	4-24
Station EDITOR Attribute	4-24
Station ENABLED Attribute	4-25
Station INITIALIZE Attribute	4-25
Station INPUTACTION Attribute	4-26

Station MYUSE Attribute	4-26
Station OUTPUTLIMIT Attribute	4-27
Station READY Attribute	4-27
Station TERMINAL Attribute	4-28
Station TITLE Attribute	4-28
STATIONSET Definition	4-29
Stationset Member List	4-30
Stationset PHONE Attribute	4-31
TERMINAL Definition	4-32
Terminal DEFAULT Specification	4-32
Terminal ADDRESSLENGTH Attribute	4-33
Terminal CLASS Attribute	4-33
Terminal LINEWIDTH Attribute	4-34
Terminal MAXINPUT Attribute	4-34
Terminal MAXOUTPUT Attribute	4-35
Terminal PAGECOUNT Attribute	4-35
Terminal PAGESIZE Attribute	4-36
Terminal RECEIVEDDELAY Attribute	4-36
Terminal SCREEN Attribute	4-37
Terminal TRANSMITDELAY Attribute	4-37
Terminal TYPE Attribute	4-38
Terminal WRAPAROUND Attribute	4-38
HARDWARE DEFINITION	4-39
Configuration Module Example	4-42

Appendix A. System Messages

Appendix B. NDLII Compiler and NDLII Post Compiler

NDLII Compiler	B-1
Compiler Files	B-1
Compiler Control Records	B-2
Compiler Control Options	B-4
CLEAR Option	B-4
CODE Option	B-5
DELETE Option	B-5
ERRORLIMIT Option	B-5
ERRORLIST Option	B-6
INCLUDE Option	B-6
LINEINFO Option	B-7
LIST Option	B-7
LIST\$ Option	B-8
LISTDELETED Option	B-8
LISTINCL Option	B-8
LISTP Option	B-8
MAP Option	B-9
MERGE Option	B-9
NEW Option	B-10
PAGE Option	B-10

PAGESIZE Option	B-10
SEQCHECK Option	B-11
SEQUENCE (SEQ) Option	B-11
Sequence Base Option	B-12
Sequence Increment Option	B-12
SUMMARY Option	B-12
TITLE Option	B-13
TRAINID Option	B-13
UPCASELISTING Option	B-14
VERSION Option	B-14
VOID Option	B-15
WARNFATAL Option	B-15
XREF Option	B-16
NDLII Post Compiler (NPC)	B-16
Configuration Overview	B-18
Running the Post Compiler	B-18
Using a WFL Job Deck	B-19
Using the CANDE <i>COMPILE</i> Command	B-19
Card File User Options	B-21

Appendix C. Reserved and Recognized Words

Reserved Words in the Protocol Module	C-1
Recognized Words in the Protocol Module	C-2
Reserved Words in the Configuration Module	C-4

Appendix D. NDLII Language Components

Recognized Compiler Control Options (\$ Options)	D-1
General Information	D-1
Protocol Module Declarations	D-3
Editor Declarations	D-3
Editor Statements	D-4
Algorithm Declarations	D-5
Line Control Declarations	D-5
Line Control Statements	D-5
Adaptor Control Declarations	D-7
Adaptor Control Statements	D-7
Configuration Module Definitions	D-8
ATTRIBUTE Definition	D-8
File Definition	D-8
GROUP Definition	D-8
LINE Definition	D-9
STATION Definition	D-9
STATIONSET Definition	D-9
TERMINAL Definition	D-10

Appendix E. NDLIANALYZER

File Names	E-1
NDLIANALYZER Options	E-2
Error Messages	E-3
Display Formats	E-5
Group Display Format	E-5
Line Display Format	E-7
Process Display Format	E-8
STATIONSET Display Format	E-10
Station Display Format	E-10

Appendix F. Understanding Railroad Diagrams

Glossary	1
Bibliography	1
Index	1

Figures

1-1.	Data Communications Information Flow	1-2
1-2.	Requests and Results	1-7
1-3.	Result/Request Communication Paths	1-8
1-4.	Line Control/Adaptor Control Communication Paths	1-9
B-1.	Compiler Files	B-1
B-2.	NDLII Post Compiler (NPC) Files	B-17
B-3.	Initializing EDCDLP or DCA	B-17
F-1.	Railroad Constraints	F-5

Tables

2-1.	Complement Operator	2-23
2-2.	Logical Operators	2-23
2-3.	Conditional Operators	2-23
2-4.	Operator Precedence	2-24
2-5.	Masking Functions	2-29
3-1.	Line Class Attributes	3-9
3-2.	BCS Type Values for Horizontal Parity Checking	3-12
B-1.	CARD File User Options	B-20
E-1.	NDLIANAYZER Error Messages	E-3

Section 1

NDLII Concepts and Structures

NDLII provides a structured, user-oriented data comm implementation language that is used on A Series and B 7900 systems to describe a data communications network physically, logically, and functionally.

The NDLII source code supplies the NDLII compiler with the information that allows the compiler to produce the proper network information file II (NIFI) that contains the code for data link processors, data structures, and information that is used to operate the data comm network. (NIFI refers to the second generation of the network information file, or NIF.) If you are using an enhanced data communications data link processor (EDCDLP) or a data communications host adapter (DCHA), and if you are using user-written protocols, you would use the NDLII post compiler (NPC) against the NIFI produced by the NDLII compiler to produce a new NIFI that contains EDCDLP or DCHA protocol code.

NDLII provides compatibility with existing system software at the MCS and Master Control Program (MCP) levels.

NDLII provides the high-level language constructs required to exploit the hardware and firmware capabilities of NSP subsystems, including bit-oriented protocols, full duplex communications, and exchange and reconfiguration features.

NDLII provides a convenient vehicle for implementing sophisticated data comm networks without requiring cumbersome programming or detailed knowledge of machine hardware.

An NDLII source program describes a data comm network physically, logically, and functionally. Physical components of a data comm network include the hardware specifications and capabilities of the various elements that make up the network. The logical characteristics of a network are the associations among the various components of the data comm subsystem (including user programs and MCSs), application-oriented characteristics (including page size and width and special-purpose characters), and the symbolic names used to reference physical elements within the network. The primary functional aspect of a network is the flow of information between a host system and terminals (including remote printers and front-end processors). Figure 1-1 illustrates the flow of information between the various components of a data comm subsystem.

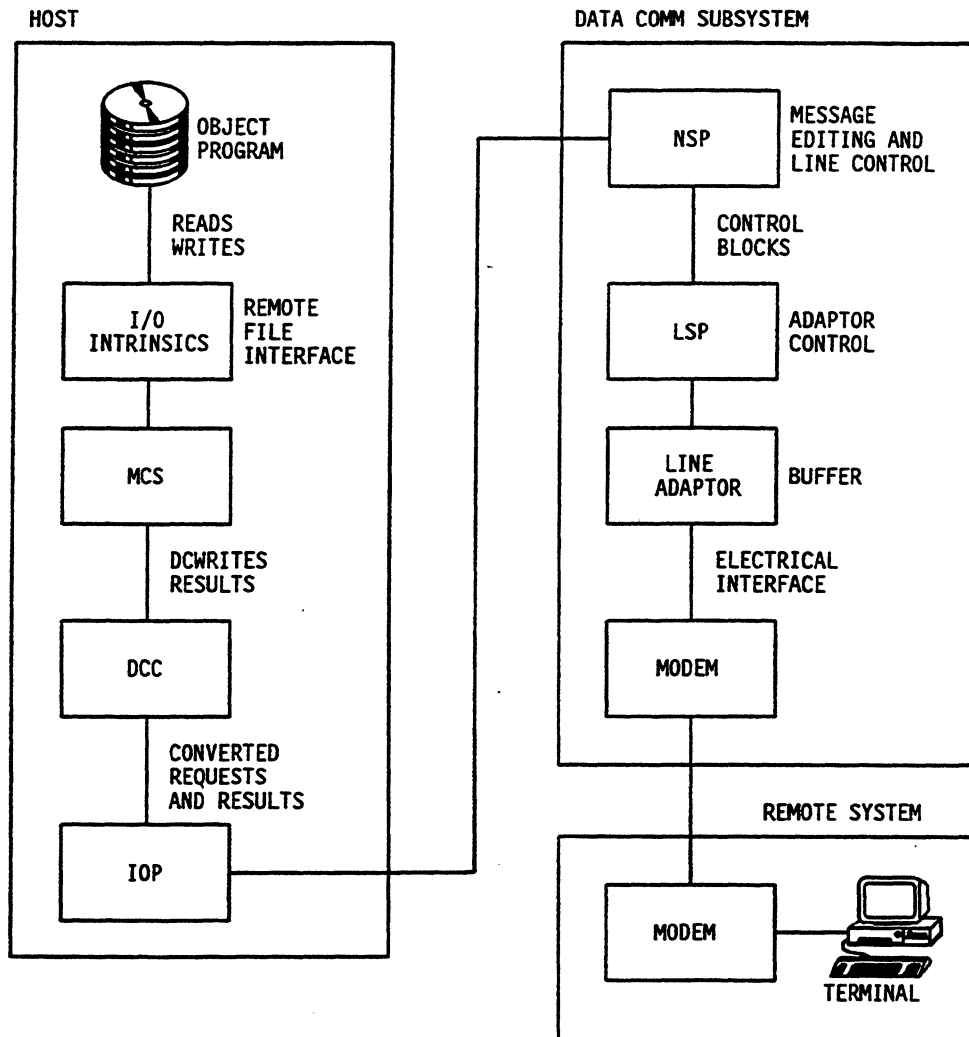


Figure 1-1. Data Communications Information Flow

NDLII Program Structure

An NDLII program consists of a protocol module and a configuration module. The protocol module contains editors and algorithms, as well as declarations for identifiers that are to be visible throughout the module. The configuration module specifies the attributes of, and relationships between, the lines, stations, and terminals in the data comm network.

Protocol Module Structure

The protocol modules defines the following three types of processes:

- Editor processes
- Line control processes
- Adaptor control processes

Editor and line control processes run in NSPs. If the system is configured without an NSP, they run in the host. Adaptor control processes run in LSPs.

Editor processes are initiated when text is moved from requests or to results; these processes are primarily used to edit the text portion of messages. Editor processes allow all editing operations to be separated from the processes that handle the line protocol itself (the line control and adaptor control processes). NDLII provides a number of powerful string-manipulation functions that can be used for editing.

The line control process and adaptor control processes for each line appear inside a single block in an NDLII program; these blocks constitute the algorithms. Each line has one associated algorithm, which handles the line protocol for that line. Algorithms can be bit-oriented (such as BDLC or SDLC) or character-oriented (such as poll/select, RJE, or TTY™). Line control processes are responsible for the higher-level aspects of the line protocols and for station management. When a transmit or receive operation on a line is required, the line control process initiates an adaptor control process. Adaptor control processes are responsible for the lower-level aspects of the line protocols and for the control of the line adaptors.

Adaptor control and editor processes do not execute continuously but are activated only under the control of line control processes.

Section 3 describes the processes that compose the protocol module.

TTY is a trademark of AT&T Teletype Corporation.

Configuration Module Structure

The configuration module defines the structure of the data comm network and the physical and logical attributes for lines and devices. These attributes are defined in Section 4. The following definitions appear in the configuration module:

- Attribute definitions
- Terminal definitions
- Station definitions
- Stationset definitions
- Line definitions
- Group definitions
- File definitions
- Hardware definitions

NDLII Source Program Example

The following example illustrates the basic structure of an NDLII source program:

```
PROTOCOL MODULE :
  [declarations]
  EDITOR <editor ID>
    BEGIN
      [declarations]
      PROCESS INPUT
        <process body>
      ENDPROCESS INPUT;
      PROCESS OUTPUT
        <process body>
      ENDPROCESS OUTPUT;
    END;
  BIT ALGORITHM <algorithm ID>
    BEGIN
      [declarations]
      LINE CONTROL
        BEGIN
          [declarations]
          PROCESS LINE
            <process body>
          ENDPROCESS LINE;
          [other process declarations]
        END;
```

```
ADAPTOR CONTROL
  BEGIN
    [declarations]
    PROCESS INPUT
      <process body>
    ENDPROCESS INPUT;
    PROCESS OUTPUT
      <process body>
    ENDPROCESS OUTPUT;
  END;
END
ENDMODULE;
CONFIGURATION MODULE :
  <attribute definition>;
  <default terminal definition>;
  <terminal definition>;
  <default station definition>;
  <station definition>;
  <stationset definition>;
  <default line definition>;
  <line definition>;
  <group definition>;
  <file definition>;
  <hardware definition>
ENDMODULE
```

SOURCENDLII

SOURCENDLII, referenced in this manual, is an NDLII source program that provides an example of how an NDLII program can be written. SOURCENDLII can be used as an example NDLII program; however, variables declared in each NDLII program should match the particular interface of the host system on which the NDLII program is to be run.

NDLII Compiler

The NDLII source program supplies the NDLII compiler with information to be loaded into the NSPs, LSPs, and DCDLPs in the network. An NDLII program consists of two functionally independent pieces of information: the NSP/LSP or DCDLP processes and the network description (the protocol module and configuration module, respectively).

Input to the compiler consists of a series of records. A record includes a data area (in columns 1 through 72) and a sequence number area (in columns 73 through 80). Records shorter than 80 characters are compiled as if blanks existed in the missing columns. The compiler does not compile information past column 80.

The data area of a particular record ends either in column 73 or at the beginning of a comment, whichever occurs first. A comment begins with the first percent sign (%) in the data area that is not part of a literal and ends at the beginning of the sequence number area.

The data areas of sequential records are interpreted as a continuous stream of input with an implied token separation between data areas. This token separation prevents tokens from being split across data area boundaries and separates a token ending in the last character position of one data area from a token beginning in the first character position in the next data area.

The compiler notifies the programmer of any syntax errors and of errors in semantics (that is, erroneous use of a syntactically correct construct) that can be detected at compile time. Semantic restrictions that cause compile-time errors when violated are described as requirements (for example, "THE <template guard> s MUST ALL BE OF THE SAME TYPE"). Restrictions that cannot be enforced at compile time are checked at run time; a violation of any of these restrictions causes the process to be fault-terminated (abnormally terminated). Any restriction that can cause a fault-termination is explicitly noted in this manual followed by the resulting error message, which is displayed to the operator (for example, "IF ENDTEXT IS TRUE, A FETCH OVERFLOW ERROR OCCURS AND THE PROCESS IS FAULT-TERMINATED"). A list of these error messages and the conditions that generate them can be found in Appendix A.

If no errors are detected by the compiler, an output file called the network information file II (NIFII) is generated. The NIFII contains the object code generated from the NDLII statements in the protocol module and the network description generated from the information in the configuration module. The items produced by the NDLII compiler are loaded during data comm initialization by the MCP. For more information on data comm initialization, refer to the *A Series Data Communications Installation and Implementation Guide*.

Appendix B describes the files used by the NDLII compiler and the compiler control records accepted by the NDLII compiler.

NDLII Post Compiler

The NDLII post compiler (NPC) is a program that generates protocols for the enhanced data communications data link processor (EDCDLP) or data communications host adapter (DCHA) from code found in the NIFII, generated by the NDLII compiler. Refer to "NDLII Compiler" in this section for information on the NIFII. You would use the NPC only if you use EDCDLPs or DCHAs in your data comm subsystem.

Because the processor and communication controllers on the EDCDLP and DCHA are similar to those used on the DCDLP, much of the EDCDLP and DCHA firmware has evolved from the DCDLP firmware. However, the existing firmware has been modified to provide an environment similar to NSP/LSPs for the protocols generated by the NPC. Thus, references to the DCDLP within this manual also apply to the EDCDLP and the DCHA, except where differences are explicitly noted.

The benefit of using the NPC is the ability to use NDLII to develop protocols for the EDCDLP and the DCHA. If you add algorithms or modify existing algorithms, you need only modify the NDLII source code, compile it, and then use the NPC.

For more information about the EDCDLP and the DCHA, refer to the *A Series Data Communications Protocols Installation and Implementation Guide*.

Appendix B describes how to run the NPC.

The Data Comm Subsystem

This information describes the logical environment in which NDLII programs execute, and the software systems required to run the data comm subsystem.

Data Comm Subsystem Overview

The host system and the data comm subsystem communicate through messages called requests and results. The host system sends requests to the subsystem and receives results in return, as shown in Figure 1-2.

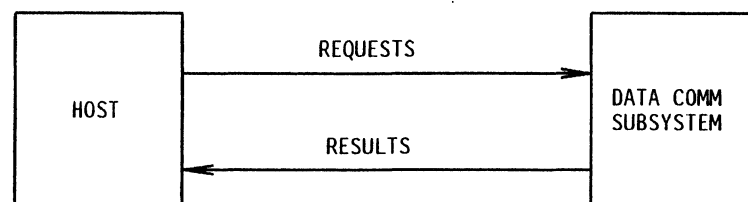


Figure 1-2. Requests and Results

Data Comm Subsystem Modules

The data comm subsystem includes three modules:

- The executive is a program provided by the system and is responsible for managing the execution of processes and the allocation of resources and for handling the host interface.
- Editors are user-provided modules written in NDLII that are used to manipulate the text portion of input and output messages.
- Algorithms are user-provided modules that include the routines necessary to perform line and station protocols, detect and recover from errors, and provide the host system with any information it might require about the status of messages and the network in addition to information provided by the executive.

Figure 1-3 shows the paths followed by results and requests within the data comm subsystem.

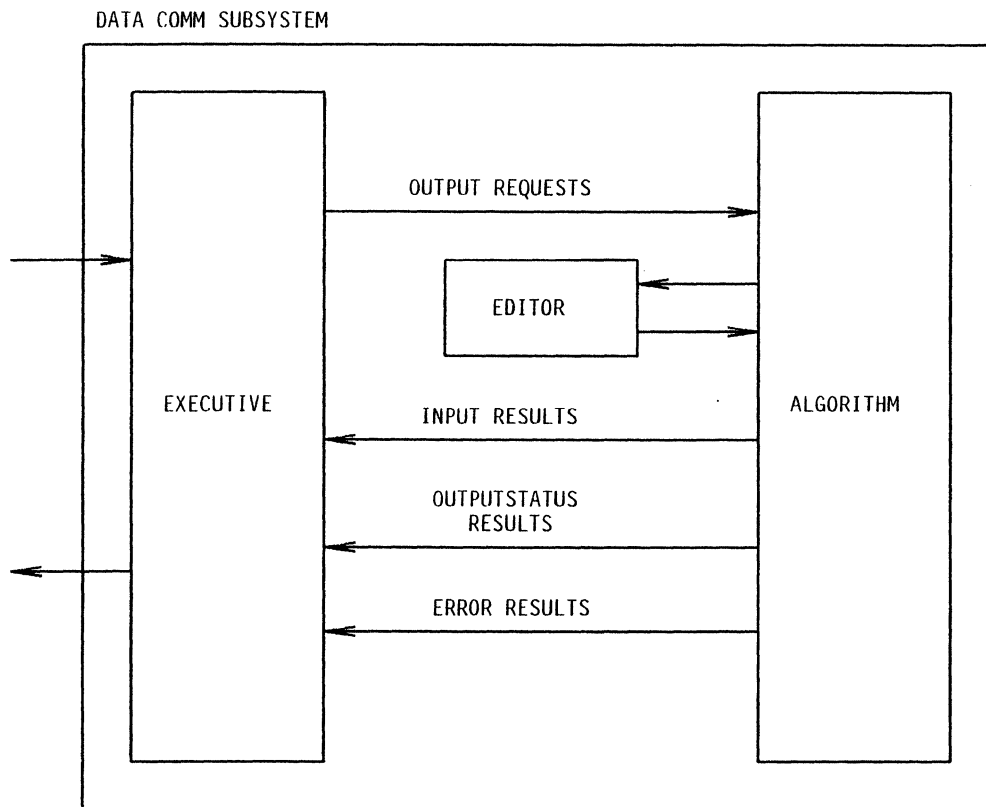


Figure 1-3. Result/Request Communication Paths

The executive directly processes all requests from the host system except output requests. Output requests are queued for subsequent action by the appropriate algorithm, which analyzes the request, invokes an output editor (if one is specified), and transmits the text. If the host system has asked that a status result be returned when the output request has been completed, the executive returns an OUTPUTSTATUS result. When an algorithm receives input from a station, it passes the data to an input editor (if one is specified) and then either returns the input text (and other relevant data) to the host system as an input or error result or discards the input.

An algorithm is composed of two distinct components: line control and adaptor control. Line control is responsible for the high-level line protocol, the generation of poll lists, the resolution of station priorities, and most error recovery. Adaptor control handles the character interface to the line on a message-by-message basis. Adaptor control runs as two independent processes, one for input and one for output.

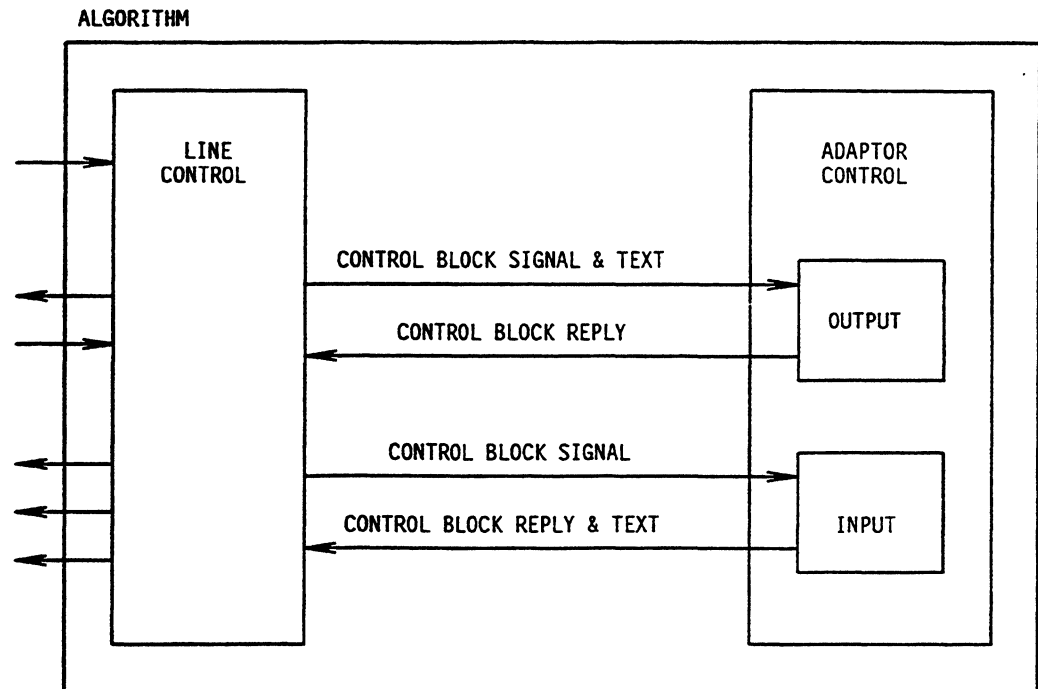


Figure 1-4. Line Control/Adaptor Control Communication Paths

Line control and adaptor control are designed to run in different physical environments; information to be transferred between the two modules is copied from one environment to the other in data structures called control blocks. Signal and reply areas can be declared as part of each control block and are used to pass information (other than the message text) back and forth.

The structure of a program written in NDLII reflects the modularity and paths of communication described previously.

Data Comm Subsystem Software Requirements

When the communications hardware has been installed on an A Series or a B 7900 system, several software systems are required in order to generate and operate the data comm network. These systems consist of the data communications controller (DCC), one or more MCSs, the NDLII compiler, and if the system includes EDCDLPs or DCHAs, the NPC. The purpose, function, and use of the DCC and an MCS are described in the following paragraphs.

Data Communications Controller (DCC)

The DCC is the basic interface between the data comm subsystem and the main system. The DCC exists as a subset of the basic A Series and B 7900 MCP and operates as a group of independent tasks or stacks, each associated with an active NSP or DCDLP.

Before you can use a defined data comm network, you must initialize the NSPs, LSPs, and DCDLPs in the network. As each one is initialized, the portion of the NDLII-defined network that uses that data link processor (DLP) becomes active.

After being initialized, each DCC stack transfers messages between the associated NSP/DCDLP and the proper MCS.

Message Control System (MCS)

An MCS is a special-purpose DCALGOL program that can be either a Unisys-supplied program (SYSTEM/CANDE, SYSTEM/APL, SYSTEM/RJE, SYSTEM/MCS, SYSTEM/DIAGNOSTICMCS, or SYSTEM/COMS) or a user-written program. The primary function of an MCS is to control the flow of data comm messages between terminals and the main system. The DCC forwards information from the data comm subsystem, such as terminal input and status information, to the primary queue of the MCS. The MCS invokes an intrinsic function called DCWRITE to forward messages such as terminal output or network changes from the MCS to the data comm subsystem.

Section 2

Common Language Components

This section describes the basic language components, general-purpose declarations, statements, and expressions that are common to both the protocol module and the configuration module. Within each of the subsections, the constructs are ordered alphabetically.

Data Types

A data type is a group of characteristics that is associated with an item in a program. Each item and its associated data type are identified by a symbolic name called an identifier. To associate each data type with an identifier, language constructs called declarations are used. Identifiers can be associated with variables, procedures, structures (collections of variables), and many other general- and special-purpose constructs.

NDLII has a number of items that have predeclared data types. These predeclared identifiers are described in the section dealing with the language environment in which the identifier is recognized.

User-declared variables are of four basic data types: Boolean, byte, integer, and string. A class of data types called enumerated types also exists. An enumerated type is represented by a set of named values called enumerated constants. Enumerated types can be either predeclared or user-declared. Variables of a particular enumerated type, known as enumerated variables, can be declared in a manner similar to Boolean, byte, integer, and string variables.

Many components of the NDLII environment (for example, stations) are created and removed dynamically, and direct access to these components cannot be provided by either predeclared or user-declared identifiers. The reference data type is used to provide indirect access to such components. The only variables of this type are predeclared and can be changed only by special statements. A reference variable might not access an object at any particular time; if not, the variable has the value NULL. The line control NULLREFERENCE function can be used to test whether or not a reference variable is NULL. For more information, refer to “NULLREFERENCE Function” in Section 3.

Language Elements

The following elementary language components are used throughout the manual to describe other NDLL constructs. Constants are described under “Constants” later in this section.

<binary digit>

One of the characters 0 or 1.

<character>

One of the 256 EBCDIC characters.

<constant>

A constant is a fixed value. A constant can be associated with an identifier to allow its value to be referenced by that identifier rather than explicitly in a program. Refer to “Constants” later in this section for a description of the types of constants available in NDLL.

<digit>

One of the characters 0 through 9.

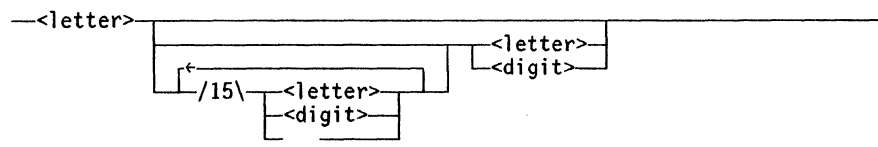
<hex character>

—<hex digit>—<hex digit>—

<hex digit>

One of the 16 characters 0 through 9 and A through F.

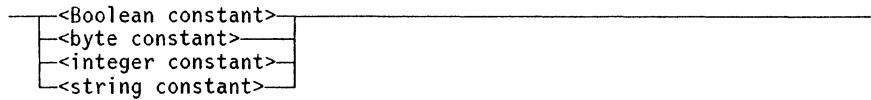
<identifier>



<letter>

One of the 52 characters A through Z and a through z. The lowercase characters (a through z) map into, and are synonymous with, the uppercase characters (A through Z).

<literal>



<octal digit>

One of the characters 0 through 7.

<recognized word>

A recognized word is an identifier that has a predefined meaning and that can be redefined by the programmer. The recognized words are listed in Appendix C.

<reserved word>

A reserved word is an identifier that has a predefined meaning that cannot be redefined by the programmer. The reserved words are listed in Appendix C.

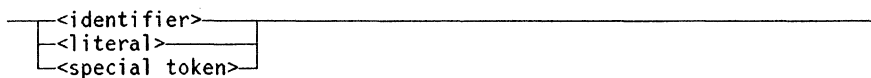
<special token>

The following items are special characters that are recognized as special tokens:

[	]	#
(	)	*
:	,	.
;	:=	=
<=	>=	^=
<	>	&
+	-	

Special tokens are symbols that are used as operators and delimiters within the language.

<token>

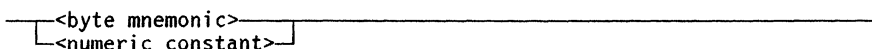


A token can be either an identifier, a literal, or a special token.

Constant values for each of the four basic data types and the enumerated constants of an enumerated type can be used in expressions and as initial values for variables of the appropriate data type. Constants of the basic data types are syntactically defined below; enumerated constants of an enumerated type are discussed under “ENUMERATION Declaration” later in this section.

FALSE
TRUE

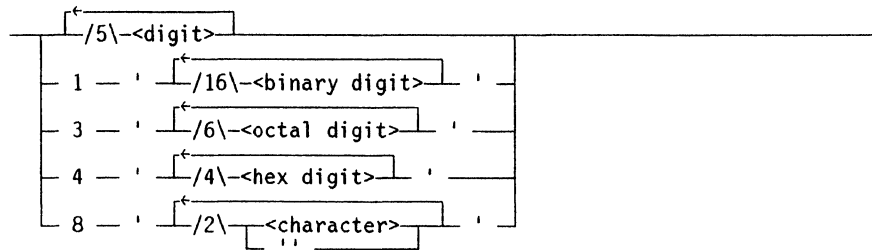
< byte constant >



< byte mnemonic >

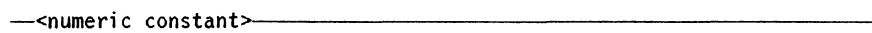
ACK = 4'2E'	ENQ = 4'2D'	POL = 4'97'
BEL = 4'2F'	EOT = 4'37'	RS = 4'1E'
BS = 4'16'	ESC = 4'27'	SEL = 4'98'
BSL = 4'A3'	ETB = 4'26'	SI = 4'ØF'
CAN = 4'18'	ETX = 4'Ø3'	SO = 4'ØE'
CR = 4'ØD'	FF = 4'ØC'	SOH = 4'Ø1'
DC1 = 4'11'	FS = 4'1C'	SP = 4'4Ø'
DC2 = 4'12'	FSL = 4'A2'	STX = 4'Ø2'
DC3 = 4'13'	GS = 4'1D'	SUB = 4'3F'
DC4 = 4'3C'	HT = 4'Ø5'	SYN = 4'32'
DEL = 4'Ø7'	LF = 4'25'	US = 4'1F'
DLE = 4'1Ø'	NAK = 4'3D'	VT = 4'ØB'
EM = 4'19'	NUL = 4'ØØ'	

<numeric constant>



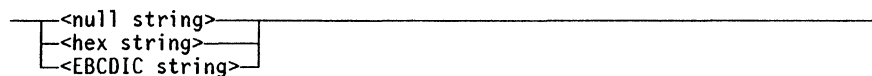
Two adjacent apostrophes (') are used to represent an apostrophe character within a numeric constant. For example, 8''' is a numeric constant representing the EBCDIC value of the apostrophe.

<integer constant>

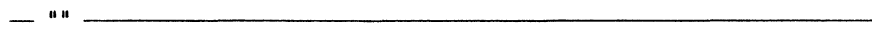


An integer constant must have a value in the range 0 through 65535 (decimal).

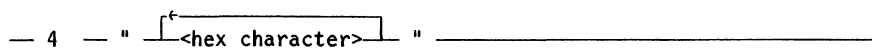
<string constant>



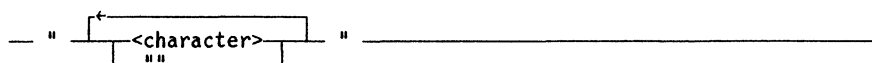
<null string>



<hex string>



<EBCDIC string>



Two adjacent double quotation marks (") are used to represent a double quotation mark character within an EBCDIC string. For example, the EBCDIC string "A""B" represents the three-character string A"B.

General Declarations

Declarations give meaning to user-provided identifiers. All identifiers that are not predeclared by the compiler must be declared before they can be used.

Identifiers that are NDLII reserved words cannot be declared (except certain process IDs that are required by NDLII). No declaration can redeclare an identifier that is already declared in the scope of that declaration (for example, an inner block cannot redeclare an identifier declared in an outer block).

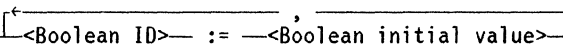
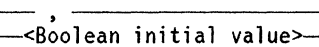
Declarations fall into two categories: general declarations and special declarations. General declarations are used for data types that are valid in more than one language environment. Special declarations are used for data types that are valid only in specific environments in the language; these declarations are described in the sections concerning each particular environment.

BOOLEAN Declaration

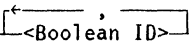
Use the BOOLEAN declaration to declare Boolean variables, which can assume the values TRUE and FALSE.

Syntax

<BOOLEAN declaration>

— BOOLEAN —  := —  —

<signal-reply BOOLEAN declaration>

— BOOLEAN —  —

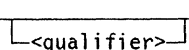
<Boolean ID>

— <identifier> —

<Boolean initial value>

— <constant Boolean expression> —

<Boolean variable>

 <Boolean ID> —

Example

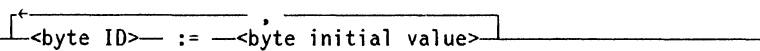
```
BOOLEAN
  DONE := FALSE,
  INPUTERROR := FALSE
```

BYTE Declaration

Use the BYTE declaration to declare byte variables. Single EBCDIC characters and integers in the range 0 to 255 can be stored in byte variables.

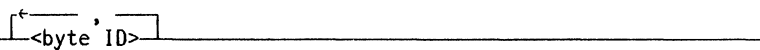
Syntax

<byte declaration>

— BYTE —  —

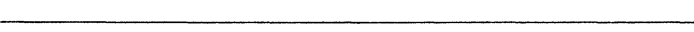
The diagram shows a horizontal line with a vertical bar at the right end. Above the line, a bracket spans from the start to a point before the colon. Above this bracket is the text '<byte ID>'. Another bracket spans from the colon to the end of the line. Above this bracket is the text '<byte initial value>'.

<signal-reply byte declaration>

— BYTE —  —

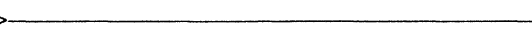
The diagram shows a horizontal line with a vertical bar at the right end. Above the line, a bracket spans from the start to a point before the colon. Above this bracket is the text '<byte ID>'.

<byte ID>

— <identifier> —  —

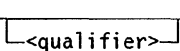
The diagram shows a horizontal line with a vertical bar at the right end. Above the line is the text '<identifier>'.

<byte initial value>

— <constant byte expression> —  —

The diagram shows a horizontal line with a vertical bar at the right end. Above the line is the text '<constant byte expression>'.

<byte variable>

 <byte ID> —  —

The diagram shows a horizontal line with a vertical bar at the right end. Above the line, a bracket spans from the start to a point before the text '<byte ID>'. Above this bracket is the text '<qualifier>'.

Example

```

BYTE
    FLAGCHAR := 8*' ',
    INDEX := 1
  
```

DEFINE Declaration

Use the DEFINE declaration to associate define text with a define ID. A DEFINE invocation is an occurrence of the define ID later in the program; at this point, the define ID is replaced by the associated define text prior to syntactic and semantic analysis.

Some analysis of the define text is performed as the declaration is processed in order to ensure that the text is structured reasonably.

Bracketing tokens in define text must match. Specifically, each left parenthesis must have a corresponding right parenthesis, each left bracket must have a corresponding right bracket, and each BEGIN must have a corresponding END.

Syntax

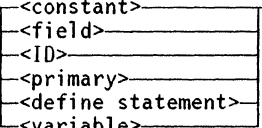
<define declaration>

— DEFINE —  <define ID> — = — # —  <define text> — # —

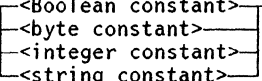
<define ID>

— <identifier> —

<define text>

—  —

<constant>

—  —

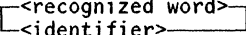
<field>

— [ <define expression token>] —

<define expression token>

Any NDLLI language token except the reserved words and two special tokens: the semicolon (;) and the number sign (#).

<ID>

—  —

<primary>

— ( <define expression token>) —

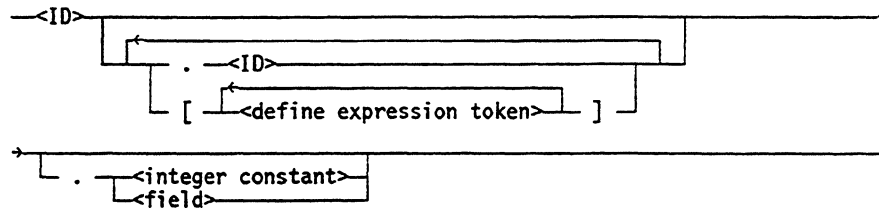
<define statement>

— BEGIN  <define statement token> — END —

< define statement token >

Any NDLLI language token except the reserved word DEFINE and the special token the number sign (#).

<variable>



Example

DEFINE

```
PRIORITYF = # POLLSTRUCTURE[I].FREQUENCY #,
LINELEN = # (72+8) #
```


ENUMERATION Declaration

Use the ENUMERATION declaration to specify the values associated with an enumerated type. An enumerated type is represented by a set of named values called enumerated constants. Variables of a particular enumerated type, called enumerated variables, are declared by an enumerated variable declaration.

Syntax

<enumeration declaration>

— ENUMERATION —<enumerated type ID> = —<enumerated value set>—

<enumerated type ID>

—<identifier>—

<enumerated value set>

—<enumerated constant list>—
—<enumerated type>—

<enumerated constant list>

— (—<enumerated constant>—
—<enumerated constant> : —<map value>—) —

<enumerated constant>

—<identifier>—

<map value>

—<constant integer expression>—

<enumerated type>

—<enumerated type ID>—
—CBCATEGORY—
—CODE EXT TYPE—
—DISCONNECTPOLICY—
—INPUTPOLICY—
—RTERROR—
—WAITRESULTTYPE—

Explanation

The enumerated constants must be unique within a particular enumerated type, and no more than 256 enumerated constants can be present within one enumerated type.

The ampersand (&) specification can be used to declare an enumerated type that is a superset (or extension) of an already-declared enumerated type. If the already-declared enumerated type is itself a superset of other enumerated types, the new enumerated type is also a superset of those enumerated types. The following example illustrates the concept of superset relationships:

ENUMERATION

```
COLOR1 = (BLUE, GREEN, RED),
COLOR2 = COLOR1 & (ORANGE),
COLOR3 = COLOR1 & (BLACK, WHITE),
COLOR4 = (YELLOW) & COLOR2,
COLOR5 = (BLUE, GREEN, RED, ORANGE, YELLOW),
COLOR6 = COLOR2 & (BLUE); % ILLEGAL
```

The enumerated types COLOR2 and COLOR3 are declared as supersets of the enumerated type COLOR1, and the enumerated type COLOR4 is declared as a superset of COLOR2. Because COLOR2 is a superset of COLOR1, COLOR4 is also a superset of COLOR1, but the enumerated type COLOR5 is not a superset of any other enumerated types even though it has exactly the same set of enumerated constants as COLOR4. These superset relations control the implicit type transfers in comparisons and assignments involving enumerated variables. The declaration of COLOR6 above is illegal because BLUE was already declared to be an enumerated constant of enumerated type COLOR2; the COLOR6 declaration has the effect of declaring BLUE twice within the same enumerated type.

Enumerated variables within requests and results are considered to be byte variables by the host system. The mapping between enumerated constants and integer values can be specified by declaring the enumerated type with the map value specification, in which case the enumerated type is said to be mapped. A map value must be an integer value in the range 0 to 255; all map values for a given enumerated type must be unique.

A mapped enumerated type can be extended (using the ampersand specification) into a nonmapped enumerated type, but a nonmapped enumerated type cannot be extended into a mapped enumerated type. If any enumerated constant in an enumerated type is nonmapped, the enumerated type is considered nonmapped.

The following enumerated types are predeclared:

- CBCATEGORY
- CODE_EXT_TYPE
- DISCONNECTPOLICY
- INPUTPOLICY
- RTERROR
- WAITRESULTTYPE

CBCATEGORY is the data type of the predeclared variable CATEGORY in control blocks. CODE_EXT_TYPE is the data type of the predeclared variables *RECEIVER.EXTEND* and *TRANSMITTER.EXTEND* in adaptor control. DISCONNECTPOLICY is the data type of the predeclared variable *LINE.DISCONNECTACTION*. INPUTPOLICY is the data type of the predeclared variable *STATION.INPUTACTION*. RTERROR is the data type of the predeclared variables *RECEIVER.FIRSTERROR* and *TRANSMITTER.FIRSTERROR* in adaptor control. WAITRESULTTYPE is the data type associated with the adaptor control function WAITFORIDLE. If these enumerated types were explicitly declared, their declarations would be as follows:

Common Language Components

ENUMERATION

CBCATEGORY = (COMPLETED:0,
INPROGRESS:1,
REJECTED:2,
ABORTED:3,
CANCELLED:4),

CODE_EXT_TYPE = (EXT_NONE:0,
EXT_SO:1,
EXT_ESC_SO:2),

DISCONNECTPOLICY = (NONE:0,
AUTOANSWER:1,
LINEDIAL:2,
STNSETDIAL:3,
STNSETDIALANSWER:4,
MANUALDIAL:5),

INPUTPOLICY = (SEND:1,
SENDANDWAIT:2),

RTERROR = (NONE:0,
DISCONNECT:1,
DATASETNOTREADY:2,
FRAMEABORT:3,
UNDERRUN:4,
OVERRUN:5,
PARITY:6,
STOPBIT:7,
TIMEOUT:8,
LOSSOFCARRIER:9,
BCSError:10,
FORMATERROR:11,
ADDRESSERROR:12,
RESIDUEERROR:13),

WAITRESULTTYPE = (TIMEOUT:0,
IDLEDETECT:1,
FRAMEDETECT:2,
ERRORDETECT:3);

Enumerated Variable Declaration

Use the enumerated variable declaration to declare variables of a particular enumerated type. Enumerated variables can assume values that are the enumerated constants of the corresponding enumerated type.

Syntax

<enumerated variable declaration>

—<enumerated type>—[←]—<enumerated variable declaration part>—|

<signal-reply enumerated declaration>

—<enumerated type>—[←]—<enumerated variable ID>—|

<enumerated variable declaration part>

—<enumerated variable ID>— := —<enumerated initial value>—|

<enumerated variable ID>

—<identifier>—|

<enumerated initial value>

—<enumerated constant>—|

<enumerated variable>

—[←]—<enumerated variable ID>—|
|<qualifier>|

Example

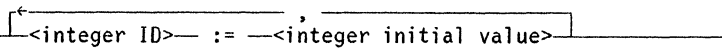
```
ERRORTYPE
    INPUTERROR := NONE,
    OUTPUTERROR := NONE
```

INTEGER Declaration

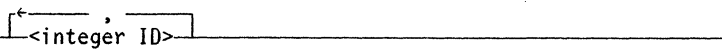
Use the INTEGER declaration to declare integer variables. Integer variables can assume integer values in the range 0 to 65535.

Syntax

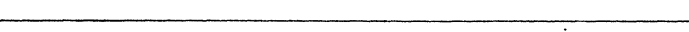
<integer declaration>

— INTEGER —  —
The diagram shows the keyword 'INTEGER' followed by a horizontal line. A bracket above this line spans from the start to just before the colon, containing the text '<integer ID>'. Another bracket above the line spans from just after the colon to the end, containing the text '<integer initial value>'. A vertical line is at the end of the main horizontal line.

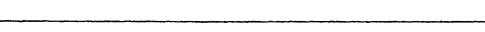
<signal-reply integer declaration>

— INTEGER —  —
The diagram shows the keyword 'INTEGER' followed by a horizontal line. A bracket above this line spans from the start to just before the comma, containing the text '<integer ID>'. A vertical line is at the end of the main horizontal line.

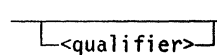
<integer ID>

—<identifier>— —
The diagram shows the text '<identifier>' followed by a horizontal line that ends with a vertical line.

<integer initial value>

—<constant integer expression>— —
The diagram shows the text '<constant integer expression>' followed by a horizontal line that ends with a vertical line.

<integer variable>

 <integer ID> —
The diagram shows a bracket above the line spanning from the start to just before the text '<integer ID>', containing the text '<qualifier>'. The line continues after the text and ends with a vertical line.

Example

```
INTEGER
  COUNT := 0,
  TIMELEFT := 4'FFFF'
```

PROCEDURE Declaration

Use the PROCEDURE declaration to declare a procedure. A procedure is a collection of statements (and local declarations) that are executed when the corresponding procedure ID appears in a call statement in the body of a process or procedure. When the procedure is exited, execution of the program continues with the statement following the call statement. PROCEDURE declarations are valid only in processes.

Syntax

```

<procedure declaration>
— PROCEDURE —<procedure ID>—<procedure body>—————→
→— ENDPROCEDURE —<procedure ID>—————|
<procedure ID>
—<identifier>—————|
<procedure body>
— BEGIN —————|
|   <procedure declaration list> — ; —————|
|   <statement> —————|
→ —————|
|   ; —————|
|   END —————|
<procedure declaration list>
—<define declaration> —————|
| <enumeration declaration> —————|
| <variable declaration list> —————|

```

Explanation

In a PROCEDURE declaration, the procedure ID that follows the word ENDPROCEDURE must be identical to the procedure ID following the word PROCEDURE.

The initial value specifications for variables declared within a procedure specify the values assigned to those variables before each invocation of the procedure.

A procedure cannot call itself.

Example

```

PROCEDURE DOSOMETHING
  BEGIN
    ...
  END
ENDPROCEDURE DOSOMETHING

```

PROCESS Declaration

Use the PROCESS declaration to declare a process. A process is a collection of statements (and local declarations) that are initiated either in response to some condition or when the corresponding process ID appears in a process statement. All executable code in NDLLI is contained within processes.

Syntax

```

<process declaration>
— PROCESS —<process ID>—<process body>— ENDPROCESS —<process ID>—|
<process ID>
—<identifier>—|
<process body>
— BEGIN —|
    |<process declaration list>— ; |<statement>— ; |
→— END —|
<process declaration list>
|<define declaration>— ; |
|<enumeration declaration>—|
|<procedure declaration>—|
|<structure declaration>—|
|<variable declaration list>—|

```

Explanation

In a PROCESS declaration, the process ID that follows the word ENDPROCESS must be identical to the process ID that follows the word PROCESS.

Certain processes are required in NDLLI and must have specific process IDs. These processes are initiated by the executive and include the LINE process of line control and the INPUT and OUTPUT processes of editors and adaptor controls. In addition to the required processes, optional processes can be declared in line control. Optional processes are initiated by a process statement. Unlike procedures, processes run in parallel with their initiators; after a process is initiated by a process statement, execution of the initiator continues with the statement that follows the process statement.

The initial value specifications for variables and structures declared within a process specify the values assigned to those variables and structures before each initiation of that process.

Example

```

PROCESS OUTPUT
  BEGIN
    ...
  END
ENDPROCESS OUTPUT

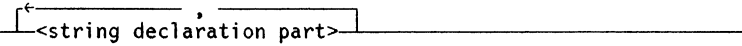
```

STRING Declaration

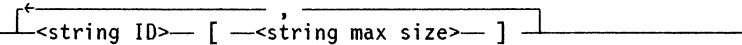
Use the STRING declaration to declare string variables. A string variable contains an ordered sequence of from 0 to 255 EBCDIC characters. You can determine the number of characters held at any time in a string variable by using the integer function LENGTH.

Syntax

<string declaration>

— STRING —  —

<signal-reply string declaration>

— STRING —  —

<string declaration part>

—<string ID>— [—<string max size>—] — := —————→

→<string initial value>—————|

<string ID>

—<identifier>—————|

<string max size>

—<constant integer expression>—————|

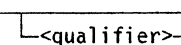
<string max size>

An integer that must be in the range 1 to 255, inclusive.

<string initial value>

—<constant string expression>—————|

<string variable>

—  <string ID> —————|

Example

```
STRING
  LASTTIME[4] := 24 & 60 & 60 & 100,
  HOSTMESSAGE[80] := "",
  WAITMESSAGE[11] := "PLEASE WAIT"
```


STRUCTURE Declaration

Use the **STRUCTURE** declaration to declare structure variables. Structures provide a way to group related variables that can be of different data types. Several structures are predeclared, including **STATION**, **LINE**, and **GROUP** in line control and **RECEIVER** and **TRANSMITTER** in adaptor control. User-declared structures can be used for a number of purposes; if declared as **DUPLICATE**, such structures can be used to communicate information between line controls and adaptor controls.

Syntax

```

<structure declaration>
— STRUCTURE —<structure part>
<structure part>
—<structure ID>—<structure array ID>— [ —<structure size>— ] —
→ ( —<structure element>— ) —
<structure ID>
—<identifier>—
<structure array ID>
—<identifier>—
<structure size>
—<constant integer expression>—
<structure element>
—<variable declaration list>—
<structure variable>
—<structure ID>—<structure array ID>— [ —<structure subscript>— ] —
<structure subscript>
—<integer expression>—
  
```

Explanation

A single structure is declared by specifying a structure ID. A structure array, which is an ordered, indexable collection of identical structure elements, is declared by specifying a structure array ID and a structure size. The number of structure elements is specified by the structure size, which must be in the range 1 to 255; the first structure element is at index 1. When accessing an element within a structure array, if the structure subscript is greater than the declared structure size or is equal to 0, a **STRUCTURE PROTECT** error occurs, and the process is fault-terminated.

A structure element is composed of Boolean, byte, enumerated, integer, and string variables. These variables are accessed by qualifying the variable identifier with the appropriate structure variable, such as *POLL/I.PRIORITY*. Initial value specifications for variables declared within the structure specify the values of those variables in all of the structure elements.

Example

```

STRUCTURE
    POLL[5] (INTEGER PRIORITY := 0;
              STRING ADDR[2] := "00")

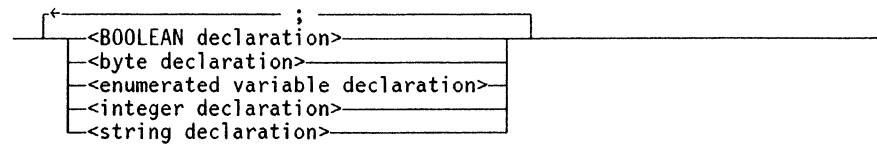
```

Variable Declaration List

BOOLEAN declarations, BYTE declarations, INTEGER declarations, and STRING declarations are used to declare variables of these basic data types. Enumerated variable declarations are used to declare variables of a particular enumerated type. Because these declarations frequently occur together in the syntax for other constructs, they are grouped here into a syntactic variable declaration list for convenience.

Syntax

```
<variable declaration list>
```



Example

```

INTEGER
    I := 0;
BYTE
    FIRSTCHAR := 8'. '

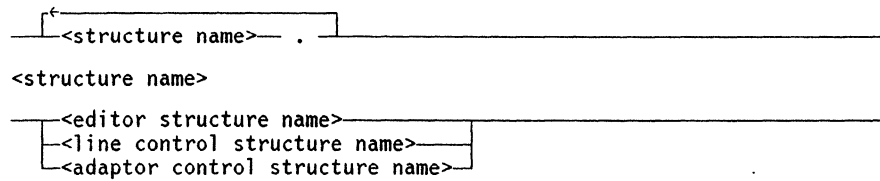
```

Qualified Variables

You can access variables that exist within user-declared or predeclared structures by qualifying the variable identifier with the name of the structure. For example, you can access the variable `TYPE` in the `STATION` structure through the qualified variable `STATION.TYPE`.

Syntax

<qualifier>



Examples

```
POLLSTRUCTURE[LSN].STATIONADDR
```

```
STATION.AVAILFLAG
```

Expressions

An expression generates a value of a particular type by performing specified operations on specified operands. The operands and operations vary according to the type of the expression.

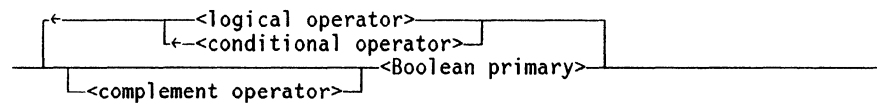
The information that follows describes the various expressions used in NDII.

Boolean Expression

A Boolean expression allows logical operations to be performed on Boolean primaries. When evaluated, a Boolean expression yields a Boolean value (either TRUE or FALSE).

Syntax

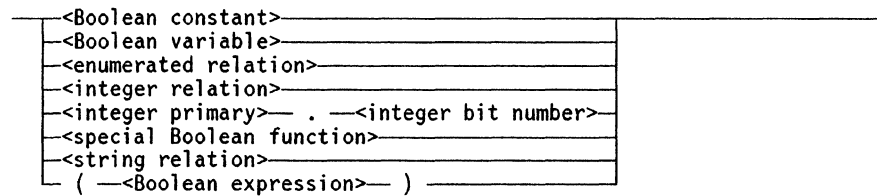
<Boolean expression>



<complement operator>

— NOT —

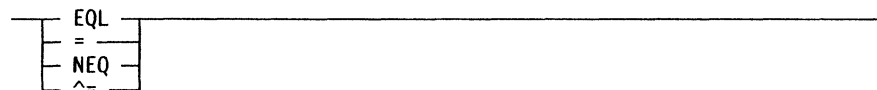
<Boolean primary>



<enumerated relation>

— <enumerated variable> <equality operator> <enumerated expression> —

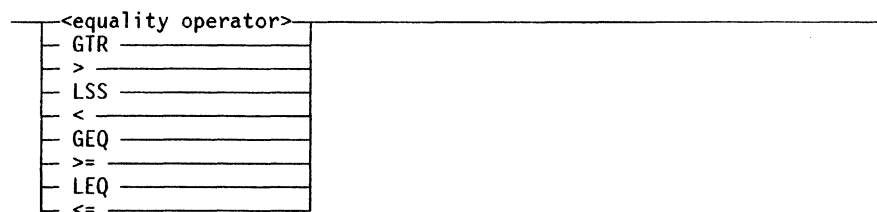
<equality operator>



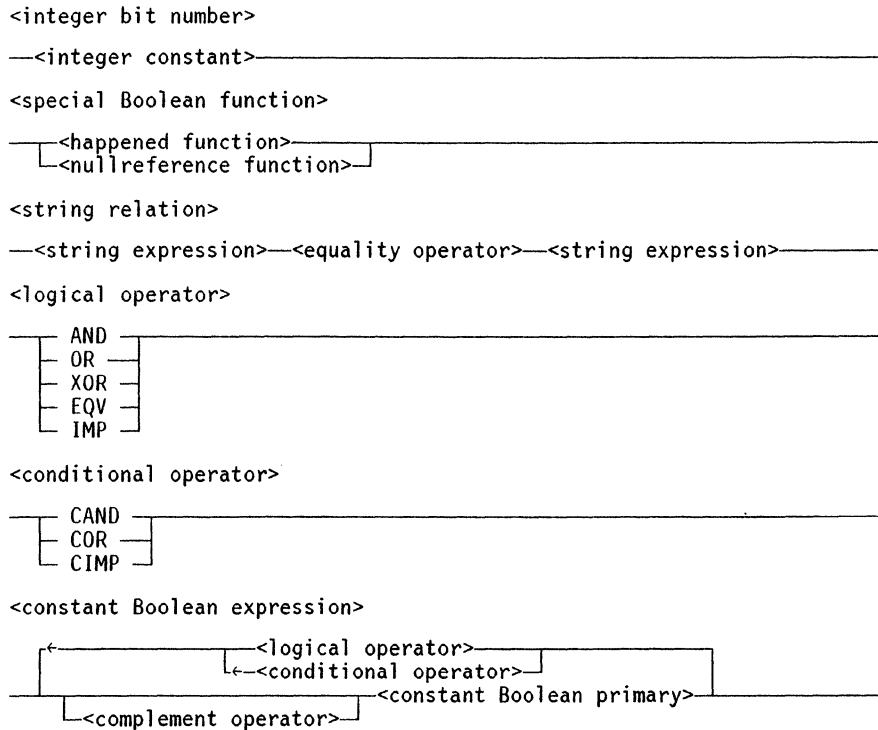
<integer relation>

— <integer expression> <relational operator> <integer expression> —

<relational operator>



Common Language Components



Explanation

A constant Boolean primary is a Boolean primary that consists entirely of constants.

An enumerated variable can be compared for equality or inequality with the value of an enumerated expression. If the enumerated expression is an enumerated constant, the constant and the variable must be of the same enumerated type. If the enumerated expression is an enumerated variable or enumerated function, either the enumerated expression and the target enumerated variable must be of the same enumerated type or the enumerated type of one must be a superset of the enumerated type of the other.

The relational operators allow comparison between two integer expressions. Comparisons are made on the entire 16-bit values of these expressions; thus, 65535 is larger than 0 (that is, 65535 is not treated as negative 1). Comparisons for values less than 0 or greater than 65535 are always FALSE.

The integer bit number construct is used to refer to a bit within an integer as a Boolean value. The value of the integer bit number must be between 1 and 16, inclusive. Bit 1 is the least significant bit of the integer quantity, and bit 16 is the most significant bit. If the selected bit in the integer primary is 0, the Boolean result is FALSE; if the selected bit is 1, the result is TRUE.

The special Boolean functions are described in Section 3 of this manual.

Two string expressions can be compared for equality or inequality. If the lengths of the strings differ, the strings are not equal. If the lengths are the same, the strings are equal only if each character in one string matches the character in the same position in the other string.

Logical operators, conditional operators, and complement operators are used to combine Boolean primaries in Boolean expressions. The action of each of these operators is described in Tables 2-1 through 2-3, followed by a discussion of precedence rules for evaluating a Boolean expression that contains more than one of these operators.

Conditional operators are similar to logical operators, except that the operand on the right is not evaluated if the value of the operand on the left is sufficient to determine the value of the operation.

Table 2-1. Complement Operator

Operand R	Operand NOT R
TRUE	FALSE
FALSE	TRUE

In Tables 2-2 and 2-3, operand is designated by the abbreviation *Opd*, and operations is designated by the abbreviation *Ops*.

Table 2-2. Logical Operators

Opd L	Opd R	Ops L AND R	Ops L OR R	Ops L XOR R	Ops L EQV R	Ops L IMP R
TRUE	TRUE	TRUE	TRUE	FALSE	TRUE	TRUE
TRUE	FALSE	FALSE	TRUE	TRUE	FALSE	FALSE
FALSE	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE
FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	TRUE

Table 2-3. Conditional Operators

Opd L	Opd R	Ops L CAND R	Ops L COR R	Ops L CIMP R
TRUE	Boolean	Boolean	TRUE	Boolean
FALSE	Boolean	FALSE	Boolean	TRUE

The precedence of logical operators, conditional operators, and the complement operator is shown in Table 2-4.

Table 2-4. Operator Precedence

Rank	Operand
First	NOT
Second	AND, CAND
Third	OR, COR
Fourth	IMP, CIMP
Fifth	XOR, EQV

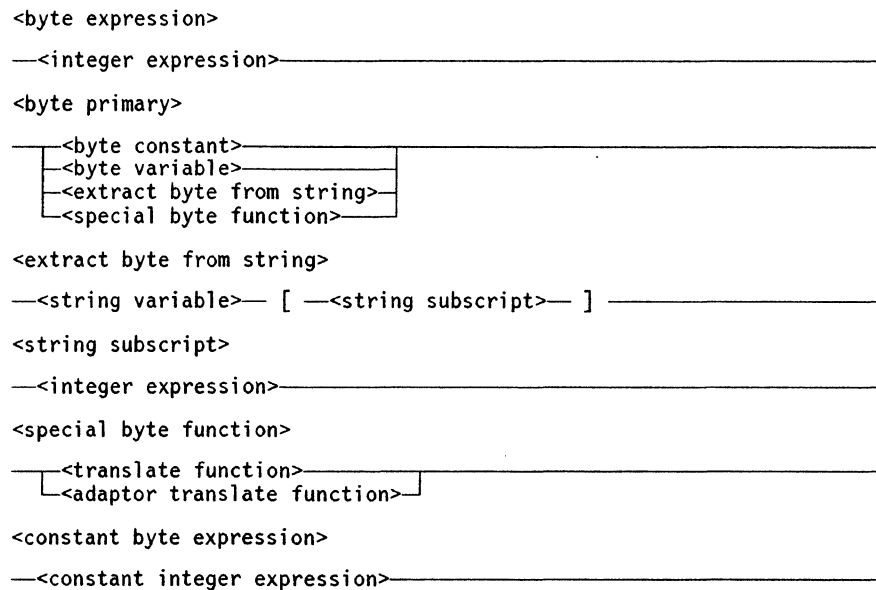
First, the unary complement operator (NOT) is applied to each Boolean primary that it precedes. Next, the AND and CAND operators are applied, followed by the other operators in the indicated order of precedence. If two consecutive operators are of the same precedence, the operators are applied in order from left to right.

Constant Boolean expressions allow logical operations to be performed on Boolean primaries that consist of all constants (constant Boolean primaries). Constant Boolean expressions can be fully evaluated at compile time.

Byte Expression

Byte expressions are formed and evaluated according to the same syntactic rules as integer expressions, except that the resulting integer value is reduced to a byte by truncating the most significant eight bits. That is, a byte expression is evaluated as if the expression *(integer expression).[8:8]* had been explicitly specified. The least significant 8 bits of the value of the integer expression are retained; the most significant 8 bits of the value are discarded.

Syntax



Explanation

Byte primaries can appear in integer expressions (and, therefore, in byte expressions) by means of byte-to-integer promotion. Refer to “Integer Expression” in this section.

Individual characters within a string variable can be accessed by subscripting the string. If the value of the string subscript is less than 1 or greater than the current length of the string, a STRING PROTECT error occurs, and the process is fault-terminated.

The translate functions are described under “Special Editor Functions” and “Adaptor Control Translate Functions” in Section 3.

Constant byte expressions are evaluated according to the same syntactic rules as constant integer expressions, except that the resulting integer constant is reduced to a byte constant by truncating the most significant 8 bits. Constant byte expressions can be fully evaluated at compile time.

Enumerated Expression

Syntax

<enumerated expression>

—<enumerated primary>—

<enumerated primary>

—<enumerated constant>—

—<enumerated variable>—

—<enumerated function>—

<enumerated function>

—<waitforidle function>—

Use the enumeration declaration to declare data types consisting of enumerated constants. An enumerated expression is an enumerated primary; no operators are defined for these types.

An enumerated constant is described in “ENUMERATION Declaration,” and an enumerated variable is described in “Enumerated Variable Declaration” in this section.

The WAITFORIDLE function is of type WAITRESULTTYPE and is described under “WAITFORIDLE Function” in Section 3.

Integer Expression

An integer expression allows arithmetic operations to be performed on integer primaries. In this section, integer constants are described under “Constants,” and integer variables are described under “INTEGER Declaration.” The remaining integer primaries are described in the material that follows.

Syntax

<integer expression>

←<arithmetic operator>
 <integer primary>

<integer primary>

<byte-to-integer promotion>
 <integer constant>
 <integer variable>
 <length function>
 <masking function>
 <max function>
 <min function>
 <rotate function>
 <shift function>
 <special integer function>
 (—<integer expression>—)

<field part>

— [—<field start>— : —<field width>—] —

<field start>

—<integer expression>—

<field width>

—<integer expression>—

<arithmetic operator>

[+]

<constant integer expression>

←<arithmetic operator>
 <constant integer primary>

< constant integer primary >

An integer primary that consists entirely of constants.

Explanation

Constant integer expressions allow arithmetic operations to be performed on integer primaries that consist of all constants (constant integer primaries). Constant integer expressions can be fully evaluated at compile time.

The field part construct creates an integer quality that contains a copy of a specific group of contiguous bits (a field) within an integer primary. The field is referenced by a field start specification, denoting the bit number of the most significant bit of the field,

and a field width specification. The least significant field width bits of the resulting integer quantity contain a copy of the specified field, and the remaining (most significant) bits are assigned zeros. If field width is not in the range 0 to field start, an INVALID OPERAND error occurs, and the process is fault-terminated. If field width is 0, no range checking on field start is performed, and the value of the integer primary is 0. If field start is not in the range 1 to 16 and field width is not 0, an INVALID OPERAND error occurs, and the process is fault-terminated.

Arithmetic operators perform addition (+) and subtraction (–) on 16-bit integer quantities. Negative numbers do not result from the actions of arithmetic operators; subtracting 1 from 0 yields 65535, and adding 1 to 65535 yields 0. No checking for arithmetic overflow or underflow is performed.

The following information discusses integer primary variables in greater detail.

Byte-to-Integer Promotion

Byte-to-integer promotion enables a byte primary to be treated as an integer primary. The resulting integer primary contains a copy of the byte primary in its least significant 8 bits and zeros in its most significant 8 bits.

Syntax

<byte-to-integer promotion>

—<byte primary>—————|

LENGTH Function

The LENGTH function returns a value equal to the current length of the value of the string expression. The current length of a string is always a value between 0 and the declared string maximum size, inclusive.

Syntax

<length function>

— LENGTH — (—<string expression>—) —————|

Masking Functions

The masking functions allow masking to be performed on all 16 bits of integer quantities.

Syntax

```
<masking function>
  ANDM ( —<integer expression>— , —<integer expression>— )
  ORM  ( —<integer expression>— , —<integer expression>— )
  XORM ( —<integer expression>— , —<integer expression>— )
  NOTM ( —<integer expression>— )
```

Explanation

Each bit position in the integer result of the function is generated by either applying the NOTM function to the corresponding bit of the parameter, or applying the ANDM, ORM, or XORM function to the corresponding bits of the two parameters. Masking functions differ from Boolean expressions in that masking functions return integer results and require integer parameters rather than returning a Boolean result from Boolean operands. Table 2-5 shows the value assigned to a result bit according to the function applied and the values of the corresponding parameter bits.

Table 2-5. Masking Functions

P1	P2	NOTM (P1)	ANDM (P1,P2)	ORM (P1,P2)	XORM (P1,P2)
0	0	1	0	0	0
0	1	1	0	1	1
1	0	0	0	1	1
1	1	0	1	1	0

MAX Function

The MAX function returns an integer quantity whose value is the larger of the values of the two integer expressions. All integer quantities are treated as 16-bit unsigned values.

Syntax

```
<max function>
  MAX ( —<integer expression>— , —<integer expression>— )
```

Example

```
LIMIT := MAX(NUMMSGs,QSIZE)
```

MIN Function

The MIN function returns an integer quantity whose value is the smaller of the values of the two integer expressions. All integer quantities are treated as 16-bit unsigned values.

Syntax

<min function>

— MIN — (—<integer expression>— , —<integer expression>—) ———|

Example

```
CHARSLEFT := MIN(SLENGTH, MAXINDEX-INDEX)
```

Rotate Function

Use the rotate function to rotate the value of the integer expression variable to the right (ROTATERIGHT) or to the left (ROTATELEFT) by the number of bits specified by the rotate count variable modulo 16. If the rotate count is 0, the result is the original value of the integer expression. Rotation to the right causes the least significant bits to be cycled to the most significant portion of the value. Rotation to the left causes the most significant bits to be cycled to the least significant portion of the value. Rotation to the right by a value C is equivalent to rotation to the left by the value $16 - (C \text{ MOD } 16)$; rotation to the left by C is equivalent to rotation to the right by $16 - (C \text{ MOD } 16)$. Byte expressions are promoted to integer expressions before the actual rotation; that is, rotation always involves 16-bit quantities.

Syntax

<rotate function>

┌ ROTATELEFT ─┐ (—<integer expression>— , —<rotate count>———>
└ ROTATERIGHT ┘

→) —————|

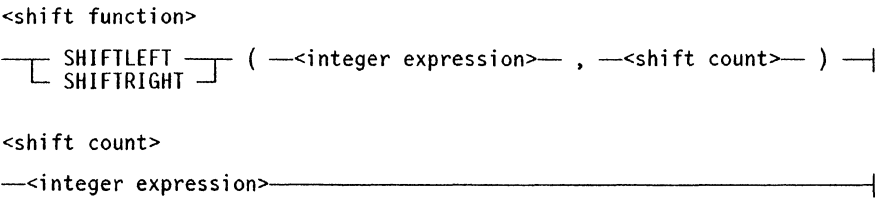
<rotate count>

—<integer expression>—————|

Shift Function

Use the shift function to shift the value of the integer expression to the left (SHIFTLEFT) or the right (SHIFTRIGHT) by the number of bits specified by the shift count. If the shift count is greater than 15, the result is 0; if the shift count is 0, the result is the original value of the integer expression. Shifting to the right causes the least significant bits of the value to be discarded; zeros are shifted to the most significant bit positions. Shifting to the left causes the most significant bits to be discarded; zeros are shifted to the least significant bit positions.

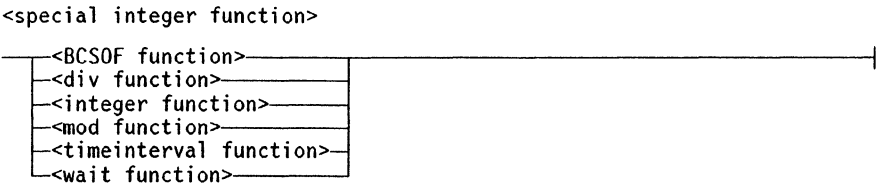
Syntax



Special Integer Functions

The BCSOF function, the DIV function, the INTEGER function, the MOD function, the TIMEINTERVAL function, and the WAIT function are described in Section 3.

Syntax



String Expression

A string expression is a string primary.

The special string functions are described in “Special Editor Functions” in Section 3. The special string primaries are described under “Line Control” in Section 3. In this section, string constants and byte constants are described under “Constants,” string variables are described in “STRING Declaration,” and byte expressions are described in “Byte Expression.”

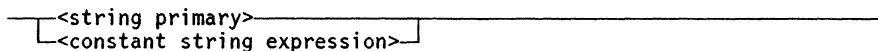
A complex string expression allows complex string primaries to be concatenated.

The concatenation operator joins two string values. The resulting string value is formed by appending a copy of the complex string primary to the right of the operator onto the end of a copy of the complex string primary to the left of the operator. The length of the resulting string is the sum of the lengths of the strings on either side of the operator. If the length of the resulting string is greater than 255, a STRING PROTECT error occurs, and the process is fault-terminated.

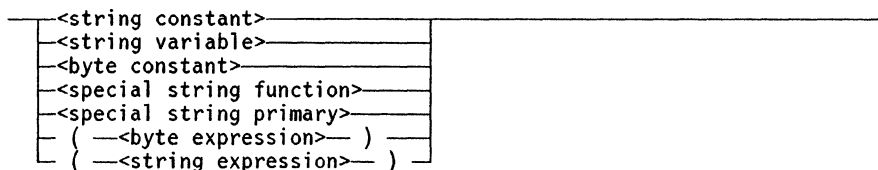
Constant string expressions allow the concatenation of string primaries that consist of all constants (constant string primaries). Constant string expressions can be fully evaluated at compile time.

Syntax

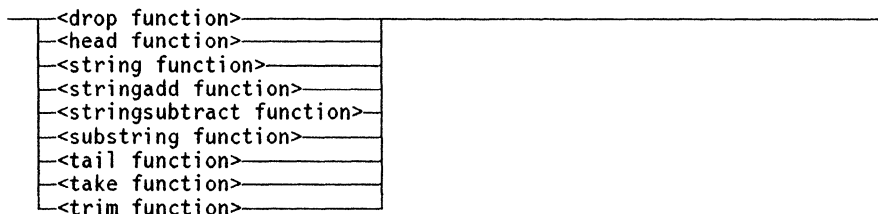
<string expression>



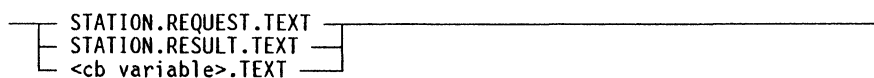
<string primary>



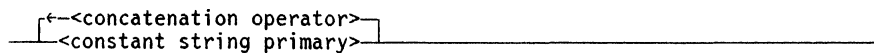
<special string function>



<special string primary>



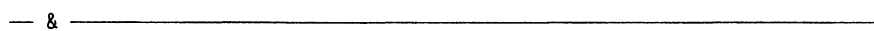
<constant string expression>



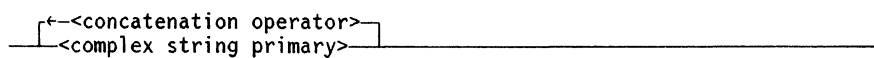
< constant string primary >

A <string primary> that consists entirely of constants.

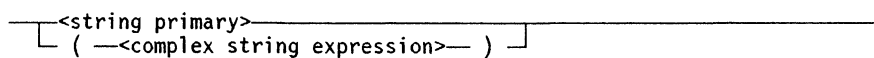
<concatenation operator>



<complex string expression>



<complex string primary>



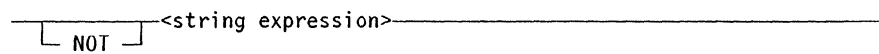
Character Set

A character set defines a set of EBCDIC characters. If NOT is not specified, the character set consists of the set of all EBCDIC characters in the value of the string expression. If NOT is specified, the character set consists of the set of all EBCDIC characters except those characters in the value of the string expression. The same character can appear more than once in the value of the string expression, but only the first appearance has any effect.

Character sets are used in the adaptor control RECEIVE statement and TRANSMIT statement and in the editor HEAD function, TAIL function, and TRIM function.

Syntax

<character set>

 <string expression>

General Statements

Statements are the executable, or active, components of programs. Simple statements perform a single operation once. Structured statements contain statements as components. Depending on the form of the structured statement, the component statements can be executed sequentially, repetitively, or conditionally.

Statements are defined in two broad categories: general statements and special statements. Special statements can appear only within specific program environments; these statements are described in the sections of this manual that discuss those environments, which include editors, line controls, and adaptor controls. General statements can appear in any environment within the protocol module; these statements are described in the material that follows.

ABORT Statement

The ABORT statement causes a fault-termination of the process in which it is executed. The abort reason must be an enumerated variable or enumerated constant of type ABORTTYPE (declared in an aborttype declaration).

If an ABORT statement is executed in an editor or line control process, an error result is sent to the host. If an ABORT statement is executed in an adaptor control process, the CATEGORY variable of the current control block is assigned the value ABORTED, and the ABORTREASON variable is assigned the specified abort reason.

Syntax

<abort statement>

— ABORT —<abort reason>—————|

<abort reason>

—|<enumerated constant>—————|
 |<enumerated variable>|

Example

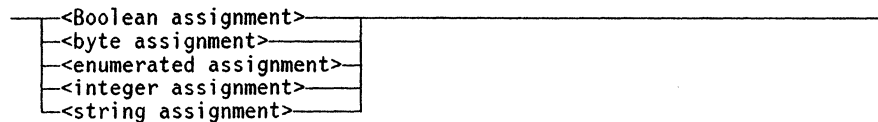
ABORT BADINDEX

Assignment Statement

An assignment statement stores the value resulting from the evaluation of an expression into a variable of the same data type. Assignment statements for each of the four basic data types and for enumerated types are described in the information that follows.

Syntax

<assignment statement>



Boolean Assignment

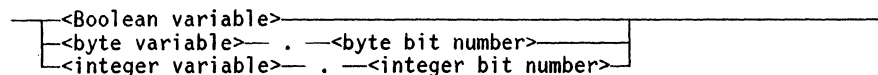
The value of a Boolean expression can be stored into a Boolean variable or into a single bit of a byte variable or an integer variable. A 0 is stored for the Boolean value FALSE, and a 1 for the Boolean value TRUE. The value of the byte bit number must be in the range 1 to 8, and the value of the integer bit number must be in the range 1 to 16. The integer bit number is described under “Boolean Expression” earlier in this section.

Syntax

<Boolean assignment>

—<Boolean target>— := —<Boolean expression>—

<Boolean target>



<byte bit number>

—<integer constant>—

Examples

```
GONE := TRUE;
```

```
FLAGBYTE.1 := DONE AND NOT FORGOTTEN
```

Byte Assignment

The value of a byte expression can be stored into a byte variable, into a field of a byte variable, into a single character of a string variable, or into a field of a single character of a string variable. If a field part is specified, the least significant field width bits of the value of the byte expression are stored in the field of the byte target specified by the field part. If the field width is 0, the value of the byte target is unchanged.

Because a byte target contains only 8 bits, numbered 1 through 8, and because a field part can specify a field outside of or overlapping this range, the following special rules apply:

1. If the difference between the field start and the field width is greater than 7, the byte target is unchanged.
2. If the field start is greater than 8 and rule 1 does not apply, then the field width is assigned the value of 8 minus the field start plus the field width, and the field start is assigned the value 8.

These rules cause any change to bits 9 through 16 to be ignored.

The field part is described in “Integer Expression” in this section.

Syntax

<byte assignment>

—<byte target> [. —<field part>] := —<byte expression>—

<byte target>

—<byte variable>
—<byte target in string>—

<byte target in string>

—<string variable> [—<string subscript>] —

Examples

HOSTMESSAGE[INDEX] := 8'.';

FIRSTCHAR.[4:4] := XCHAR.[8:4]

Enumerated Assignment

The value of an enumerated expression can be stored only into an enumerated variable. If the enumerated expression is an enumerated constant, the constant and the target variable must both be of the same enumerated type. If the enumerated expression is an enumerated variable or an enumerated function, either the enumerated expression and the target variable must be of the same enumerated type or the enumerated type of the target variable must be a superset of the enumerated type of the enumerated expression.

Syntax

<enumerated assignment>

—<enumerated variable> := —<enumerated expression>—

Example

ERRORF := BADINDEX

Integer Assignment

The value of an integer expression can be stored into an integer variable or into a field of an integer variable. If a field part is specified, the least significant field width bits of the value of the integer expression are stored in the field of the integer variable specified by the field part. If the field width is 0, the value of the integer variable is unchanged. The field part is described in "Integer Expression" in this section.

Syntax

<integer assignment>

—<integer variable>—
└── . —<field part> ┘ := —<integer expression>—|

Examples

```
INDEX := INDEX + OFFSET;
```

```
WAITLIMIT.[16:4] := 0
```

String Assignment

A complex string expression can be stored only into a string variable. If the value of the complex string expression is longer than the declared string maximum size for the string variable, a STRING PROTECT error occurs, and the process is fault-terminated.

Syntax

<string assignment>

—<string variable>— := —<complex string expression>—————|

Example

```
HOSTMESSAGE := "STATION " & STATIONADDR & " NOT READY"
```

CALL Statement

The CALL statement causes execution control to be transferred to the procedure body associated with the specified procedure ID. When that procedure finishes execution, execution control resumes at the statement following the CALL statement.

Syntax

<call statement>

— CALL —<procedure ID>—————|

Example

```
CALL ENDITALL
```

CASE Statement

The CASE statement allows the selective execution of one of a group of statements, depending on the value of the specified case selector. The statement is executed for the Boolean expression where *<case selector> = <case guard>* has the value TRUE.

Syntax

```

<case statement>
— CASE —<case selector>— BEGIN —<case body>— END —————|
                                     | ; |
                                     |
<case selector>
—<byte expression>—————|
|<enumerated function>—|
|<enumerated variable>—|
|<integer expression>—|
|<string expression>—|
<case body>
—<case guard> : <statement>—————|
|<case guard> : <statement>—————|
|<case guard> : <statement>—————|
| ; — ELSE — : —<statement>—————|
<case guard>
—<constant byte expression>—————|
|<enumerated constant>—————|
|<constant integer expression>—|
|<constant string expression>—|

```

Explanation

For a given case statement, all case guards must be of the same type (such as all constant byte expressions). In addition, the case guards must be of a type that is compatible with the case selector. If a byte expression is used as a case selector, the case guards must be either constant byte expressions or constant integer expressions. If the case selector is an enumerated function or enumerated variable, all case guards must be enumerated constants that are specified as values for the enumerated type of the enumerated function or enumerated variable. If the case selector is an integer expression, the case guards must all be constant integer expressions. If the case selector is a string expression, all case guards must be constant string expressions.

The case guards specified in a case statement need not include all possible values of the case selector. If no case guard is specified for which *<case selector> = <case guard>* is TRUE, the statement that follows the *ELSE*: guard, if present, is executed; if the *ELSE*: guard is not present, an INVALID CASE VALUE error occurs, and the process is fault-terminated.

All case guards must be unique; that is, two or more case guards within a single case statement cannot have the same value. The case guards need not be in any particular order. The maximum value of a constant integer expression used as a case guard is 255.

Example

```
CASE I
BEGIN
  3: SKIP;    % NULL STATEMENT
  5:
    BEGIN
      I := I + 2;
      X := I - 2;
    END;
  2:
  4: ABORT BADINDEX;
  ELSE: Z := 3;
END;
```

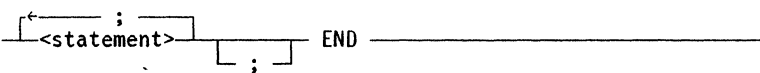
In the example above, if the value of *I* is 3, then the *skip* statement is executed. If the value of *I* is 5, then the compound statement associated with the case guard of 5 is executed. If the value of *I* is either 2 or 4, then the abort statement is executed. If *I* has any value other than 2, 3, 4, or 5, the statement following the *ELSE*: guard is executed.

Compound Statement

The compound statement allows several statements to be executed as a single group. Compound statements are frequently used as the statement within *IF* statements, *DO* statements, *WHILE* statements, and *CASE* statements.

Syntax

<compound statement>

— BEGIN —  —

Example

```
BEGIN
INDEX := 0;
CALL INITSTRINGS;
END
```

DO Statement

The DO statement allows the specified statement to be executed repeatedly until the value of the Boolean expression is TRUE. The statement is always executed at least once, even if the Boolean expression is TRUE before execution of the DO statement.

Syntax

<do statement>

— DO —<statement>— UNTIL —<Boolean expression>—————|

Example

```
DO
  BEGIN
    I := I + 1;
    CALL SENDIT;
  END
UNTIL I GTR NUMMSGs
```


IF Statement

The IF statement allows the selective execution of one of two statements, depending on the value of the specified Boolean expression. If the value of the Boolean expression is TRUE, the statement following the word THEN is executed and the statement in the optional ELSE statement clause is not executed. If the value of the Boolean expression is FALSE, the statement following the word THEN is not executed; if an ELSE statement clause appears, that statement is executed. If the Boolean expression is FALSE and no ELSE statement clause appears, program execution continues with the statement following the IF statement.

Syntax

<if statement>

```
— IF —<Boolean expression>— THEN —<statement>————→  
→ | ELSE —<statement>— |
```

Example

```
IF A LSS B THEN  
  BEGIN  
    NUMTOSEND := B;  
    FLAG := TRUE;  
  END  
ELSE  
  NUMTOSEND := A
```

SHORTEN Statement

The SHORTEN statement deletes the number of characters specified by the value of the integer expression from the end of the specified string variable. If the value of the integer expression is less than the length of the string variable, the string variable is assigned the value of the following expression:

```
TAKE(<string variable>,  
      LENGTH(<string variable>) - <integer expression>)
```

If the value of the integer expression is greater than or equal to the length of the string variable, the string variable is assigned the null string.

Although other string operations are designed to fault-terminate a process for attempting to shorten a string by more than its current length, the SHORTEN statement is designed to return the null string in this case, so that multiple backspace characters in adaptor control processes can be handled more easily.

Syntax

<shorten statement>

— SHORTEN — (—<string variable>— , —<integer expression>—) —|

Example

```
SHORTEN(HOSTMESSAGE,LEN-1)
```

SKIP Statement

The SKIP statement causes no action to be performed but is provided so that “no operation” can be specified as a statement in a CASE statement or as the statement following the word THEN or ELSE in an IF statement.

Syntax

<skip statement>

— SKIP —————|

WHILE Statement

The WHILE statement allows the specified statement to be repeatedly executed until the value of the Boolean expression is FALSE. If the initial value of the Boolean expression is FALSE, the statement does not execute.

Syntax

<while statement>

— WHILE —<Boolean expression>— DO —<statement>—————|

Example

```
WHILE I LEQ NUMMSGs DO
  BEGIN
    I := I + 1;
    CALL SENDIT;
  END
```

Section 3

Protocol Module

The constructs described in this section can be used only in the protocol module. General-purpose constructs that can also be used in the protocol module are described in Section 2. Information in this section is presented in the following order:

- Declaration or definition of an environment (such as an editor)
- Predeclared variables
- Special declarations
- Special statements
- Special functions
- Fault-termination in the particular environment

Constructs within each subsection are presented alphabetically.

Three types of processes are defined in the protocol module:

- Editor processes
- Line control processes
- Adaptor control processes

Editor processes allow all editing operations to be separated from the processes that handle the line protocol itself (the line control and the adaptor control processes).

The line control process and the adaptor control process for each line appear inside a single block in an NDLLI program; these blocks constitute the algorithms. Each line has one associated algorithm, which handles the line protocol for that line. Line control processes are responsible for the higher-level aspects of the line protocols and for station management.

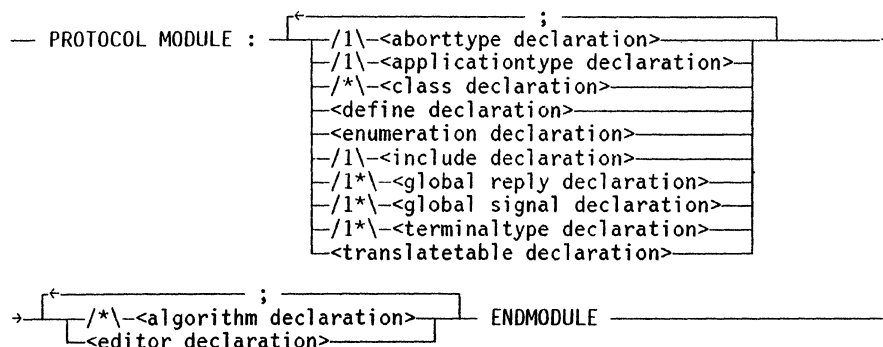
An adaptor control process is initiated by a line control process when a transmit or receive operation on a line is required. Adaptor control processes are responsible for the lower-level aspects of the line protocols and for the control of the line adaptors.

Declarations in the Protocol Module

Declarations give meaning to user-provided identifiers. All identifiers that are not predeclared by the compiler must be declared before they can be used. The protocol module uses both general and special declarations. General declarations are defined in Section 2. This subsection defines declarations that are special to the protocol module.

Syntax

<protocol module>



The protocol module can include any of the declarations listed in the syntax diagram that follows. The DEFINE declaration and the ENUMERATION declaration are described in Section 2. The ABORTTYPE declaration, the APPLICATIONTYPE declaration, the CLASS declaration, the INCLUDE declaration, the global REPLY declaration, the global SIGNAL declaration, the TERMINALTYPE declaration, and the TRANSLATETABLE declaration are described in the following text.

ABORTTYPE Declaration

The ABORTTYPE declaration is an ENUMERATION declaration that establishes the values to be allowed as abort reasons in the abort statement or as values for the control block variable ABORTREASON.

Example

```
ENUMERATION
  ABORTTYPE = (BADINDEX:0,
               INVALIDSIGNAL:1)
```

APPLICATIONTYPE Declaration

The APPLICATIONTYPE declaration is an ENUMERATION declaration that establishes the values for variables of type APPLICATIONTYPE. The enumerated constants of this enumeration are the allowed values for application keys in the station APPLICATIONLIST attribute.

An APPLICATIONTYPE declaration must appear if a station APPLICATIONLIST attribute is specified in any station definition in the configuration module.

Example

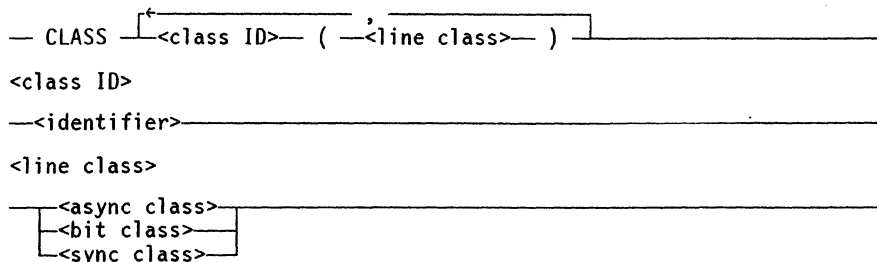
```
ENUMERATION
  APPLICATIONTYPE = (CANDEEDITOR:1,
                    CANDETAPEEDITOR:2,
                    APLEEDITOR:3,
                    RJEEDITOR:4)
```

CLASS Declaration

A CLASS declaration is used to specify line classes. A line class is a combination of a transmission mode and the values of a related set of attributes. A line class is specified for each line and terminal defined in the configuration module and can be changed by the line control INITIATE statement. The transmission modes and associated attributes are discussed under "Asynchronous Mode," "Bit Mode," "Synchronous Mode," and "Line Class Attributes" in this section.

Syntax

<class declaration>



Explanation

The following paragraphs discuss aspects of receiver and transmitter operation that apply to all transmission modes. References to statements and predeclared variables are found in "Adaptor Control" in this section.

The receiver accepts a serial stream of bits from the data comm equipment and converts the bit stream to a sequence of characters. The rate at which bits are expected is determined by the bit rate attribute in asynchronous mode, by either the data set or the bit rate attribute in bit mode, and by the data set in synchronous mode.

The receiver is enabled by an ENABLE RECEIVER statement, and each accumulated character is placed in a holding register. If the holding register is not unloaded by a RECEIVE statement before another character is accumulated, the new character is placed in the holding register and RECEIVER.ERROR.OVERRUN is assigned the value TRUE.

The number of bits assembled per character is determined by the character size of the translate table specified by the code attribute (except for certain fields in bit mode, as described under "Bit Mode"). The assembled character is right-justified with zero fill.

The transmitter accepts a sequence of commands and characters; this sequence results in a serial bit stream that is transmitted to the data comm equipment. The rate at which bits are transmitted is determined by the bit rate attribute in asynchronous mode, by either the data set or the bit rate attribute in bit mode, and by the data set in synchronous mode.

The transmitter is enabled by an ENABLE TRANSMITTER statement, and each character to be transmitted is loaded into a holding register as the result of a TRANSMIT statement. When the transmitter transfers the contents of the holding

register to the shift register, the next character must be loaded into the holding register within one character time; otherwise, an underrun condition occurs. Underrun is an error only in bit mode.

The number of bits transmitted per character is determined by the character size of the translate table specified by the code attribute (except for some fields in bit mode, as discussed under “Bit Mode”). These bits must be right-justified within the character.

Asynchronous Mode

In asynchronous transmission mode, line synchronization is maintained by framing each character with start and stop bits. Asynchronous mode is used for low- to medium-speed transmission of character strings.

Syntax

`<async class>`

```

— MODE — = — ASYNC — ; — [ /1*\-<bit rate> ;
                           | /1*\-<code>
                           | /1*\-<parity>
                           | /1*\-<stop bits>

```

Explanation

When the receiver is initiated in asynchronous mode, it recognizes the first start bit following a stop bit and then begins assembling a character, least-significant bit first. All start, stop, and parity bits are deleted from the assembled character. If the final bit of a character is not followed by a stop bit, `RECEIVER.ERROR.STOPBIT` is assigned the value `TRUE`; the absence of a stop bit in conjunction with a received character of all zeros causes `RECEIVER.ERROR.FRAMEABORT` to be assigned the value `TRUE` as well. If the `PARITY` attribute specifies even or odd vertical parity and the assembled character has bad parity, `RECEIVER.ERROR.PARITY` is assigned the value `TRUE`.

The transmitter converts each character loaded into the holding register into the following bit sequence:

- A start bit
- The bit-serial character (least significant bit first)
- A parity bit (if the `PARITY` attribute specifies even or odd vertical parity)
- The number of stop bits specified by the stop bits attribute

If the next character has already been loaded when the current character has completed, conversion of the next character is begun immediately; otherwise, the line is held in a mark condition.

If a `TRANSMIT BREAK` statement is executed, the transmitter holds the line in a space condition until either another transmit statement is executed, or the transmitter is disabled as the result of a `DISABLE TRANSMITTER` statement.

Example

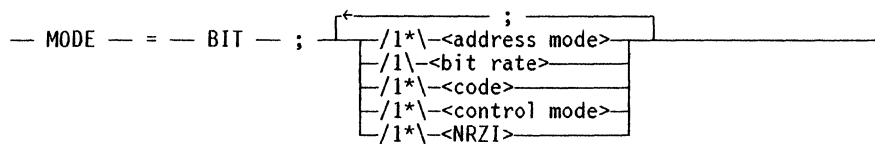
```
CLASS ASYNC110 (MODE = ASYNC;
                CODE = ASCII;
                BITRATE = 110;
                STOPBITS = LONG;
                PARITY = VERTICAL: ODD, HORIZONTAL: BCCEVEN)
```

Bit Mode

In bit transmission mode, line synchronization is maintained by operating all data comm equipment on the line at the same frequency and by keeping the data comm equipment in phase by framing each transmission with flag patterns. Bit mode is used for low- to high-speed transmission of arbitrary bit strings.

Syntax

<bit class>



Explanation

When the receiver is enabled in bit mode, it begins searching for a flag pattern (4"7E"). When a flag pattern is detected and the next 8 bits are either another flag pattern or an abort pattern (4"FF"), the search is reinitiated. Otherwise, the beginning of a frame has been detected.

The receiver continues to assemble 8-bit characters until the address and control fields of the frame have been received. Subsequent received bits are assembled based on the character size of the translate table specified by the code attribute until either a flag pattern or an abort pattern is detected. If the address mode attribute specifies BASIC, the address field consists of 1 character; otherwise, the address field ends with the first assembled character in which the low-order bit is 1. If the control mode attribute specifies BASIC, the control field consists of 1 character; otherwise, the control field consists of 2 characters. If the line function attribute specifies SECONDARY, the receiver compares the first received address character with both the global address (4"FF") and the specified select address. If a match occurs (or if the line function attribute was specified as PRIMARY), the receiver accepts the frame; otherwise, the receiver skips the frame and begins searching for the next frame.

If the frame ends with a flag, RECEIVER.RESIDUE is assigned a value that indicates the number of residue bits in the l-field, and the frame check sequence (FCS) is checked and discarded. If the final computed FCS is not correct, RECEIVER.ERROR.BCSError is assigned the value TRUE. If the frame is terminated by an abort, RECEIVER.ERROR.FRAMEABORT is assigned the value TRUE.

Within a frame, a zero-bit following five consecutive one-bits is discarded. When the transmitter is enabled, it starts to transmit continuous flags.

If a TRANSMIT FRAME statement is executed, the transmitter transmits loaded characters until either it is given an end-of-frame indication or an underrun occurs. The address and control fields are transmitted at 8 bits per character. I-field characters are normally transmitted based on the character size of the translate table specified by the code attribute, but an option in the TRANSMIT FRAME statement can force I-field characters to be transmitted at 8 bits per character. If the transmitter is given an end-of-frame indication, the transmitter performs as follows:

- If TRANSMITTER.RESIDUE is not 0 and the frame has an I-field, it transmits the specified number of residue bits from the last I-field character.
- It transmits the FCS.
- It transmits at least one flag.

If an underrun occurs, the transmitter transmits an abort followed by continuous flags and assigns the value TRUE to *TRANSMITTER.ERROR.UNDERRUN*.

Within a frame, a zero-bit is inserted after five consecutive one-bits.

If a TRANSMIT FLAGS statement is executed, the transmitter transmits continuous flags until either another TRANSMIT statement is executed, or the transmitter is disabled as the result of a DISABLE TRANSMITTER statement.

If a TRANSMIT ONES statement is executed, the transmitter transmits continuous one-bits until either another TRANSMIT statement is executed, or the transmitter is disabled as the result of a DISABLE TRANSMITTER statement.

Example

```
CLASS BITMODE (MODE = BIT;  
                CODE = BINARY;  
                NRZI = TRUE;  
                CONTROLMODE = BASIC;  
                ADDRESSMODE = BASIC)
```

Synchronous Mode

In synchronous transmission mode, line synchronization is maintained by operating all data comm equipment on the line at the same frequency and keeping this equipment in phase by starting each transmission with the character sequence SYN SYN and by embedding one of the character sequences SYN or DLE SYN in the transmission. Synchronous mode is used for low- to high-speed transmission of character sequences.

Syntax

<sync class>

```

— MODE — = — SYNC — ; {
    /1*\-<code>
    /1*\-<diagnostic rate>
    /1*\-<parity>
    /1*\-<sync control>
}

```

Explanation

When the receiver is enabled in synchronous mode, it begins searching for two contiguous characters that match the SYN character specified by the SYNCCONTROL attribute. When the receiver recognizes these characters, line synchronization is established. The receiver then begins assembling characters from the line, least significant bits first. If RECEIVER.TRANSPARENT is FALSE and the PARITY attribute specifies even or odd vertical parity, each character is assumed to include a parity bit. If a character has bad parity, RECEIVER.ERROR.PARITY is assigned the value TRUE, and the parity bit is deleted from the assembled character.

The transmitter enters nontransparent mode when it is enabled and transmits continuous SYN characters (as specified by the SYNCCONTROL attribute) until a character is loaded into the holding register. At least four SYN characters are transmitted.

Each loaded character is converted to the following bit sequence: the bit-serial character (least significant bit first), followed by a parity bit in nontransparent mode when the parity attribute specifies even or odd vertical parity. If FORCE DLE is specified by a TRANSMIT statement and TRANSMITTER.TRANSPARENT is TRUE, the transmitter does the following:

- Enters transparent mode if it is currently in nontransparent mode and transmits a DLE character (as specified by the SYNCCONTROL attribute)
- Transmits the loaded character

If an underrun occurs in nontransparent mode, the transmitter transmits the SYN character. If an underrun occurs in transparent mode, the transmitter transmits the DLE character followed by the SYN character.

Example

```

CLASS SYNCASCII (MODE = SYNC;
    DIAGNOSTICRATE = 19200;
    CODE = ASCII;
    SYNCCONTROL = DLE: 4'10', SYN: 4'16';
    PARITY = VERTICAL: ODD,
        HORIZONTAL: CRC=X7+1,0,0)

```

Line Class Attributes

The attributes for asynchronous-, bit-, and synchronous-mode line classes are described in the information that follows.

Table 3-1 shows the various line class attributes and their applicable modes.

Table 3–1. Line Class Attributes

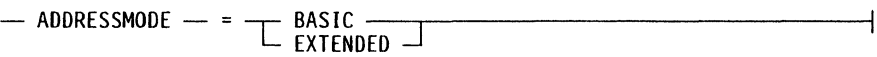
Attribute	Mode
ADDRESSMODE	BIT
BITRATE	BIT, ASYNC
CODE	BIT, ASYNC, SYNC
CONTROLMODE	BIT
DIAGNOSTICRATE	SYNC
NRZI	BIT
PARITY	ASYNC, SYNC
STOPBITS	ASYNC
SYNCCONTROL	SYNC

ADDRESSMODE (BIT Only)

The ADDRESSMODE attribute specifies the format of the address field of a bit-oriented frame. A value of BASIC indicates a 1-byte address field; a value of EXTENDED indicates that the address field ends with the first address byte that has a least significant bit of 1.

Syntax

<address mode>



BITRATE (ASYNC and BIT Only)

The BITRATE attribute specifies the rate at which bits are expected by the receiver and sent by the transmitter when the line does not use a data set. The following are valid bit rate values:

45.5	300	4800
50	600	7200
56.9	1200	9600
75	1800	19200
110	2000	38400
134.5	2400	
150	3600	

The 38400-bit rate is valid only for DCDLPs.

Syntax

<bit rate>

— BITRATE — = — <bit rate value> —————|

CODE (All)

The CODE attribute specifies the code translation to be performed by the data comm subsystem. A translate table is described under “TRANSLATETABLE Declaration” in this section.

In some situations, where special translate tables are necessary or where the translation cannot be known in advance (such as for dial-up lines), the CODE attribute can be specified as DYNAMIC and the INITIATE statement used to load a translate table dynamically. If the CODE attribute is specified as DYNAMIC, the initial translate table loaded is BINARY.

If a line has more than one station, all stations on the line must use the same translate table.

Syntax

<code>

— CODE — = — DYNAMIC —————|
 └─<translatetable>—┘

CONTROLMODE (BIT Only)

The CONTROLMODE attribute specifies the format of the control field of a bit-mode frame. A value of BASIC specifies a 1-byte control field; a value of EXTENDED specifies a 2-byte control field.

Syntax

<control mode>

— CONTROLMODE — = ☐ BASIC ☐ EXTENDED

DIAGNOSTICRATE (SYNC Only)

The DIAGNOSTICRATE attribute specifies the rate at which bits are expected by the receiver and sent by the transmitter when the line is placed in diagnostic mode. The valid bit rate values are as follows:

45.5	300	4800
50	600	7200
56.9	1200	9600
75	1800	19200
110	2000	38400
134.5	2400	
150	3600	

The 38400-bit rate is valid only for DCDLPs.

Syntax

<diagnostic rate>

— DIAGNOSTICRATE — = —<bit rate value>—

NRZI (BIT Only)

The NRZI attribute specifies whether or not the transmitter and receiver are to use NRZI (nonreturn to zero inverted) encoding. NRZI encoding in conjunction with the bit rate attribute allows bit-mode lines to run without data sets.

Syntax

<NRZI>

— NRZI — = ☐ TRUE ☐ FALSE

PARITY (ASYNC and SYNC Only)

The PARITY attribute defines the type of parity checking to be performed by the data comm subsystem. Parity types are NONE, VERTICAL, and HORIZONTAL.

A value of NONE indicates that no parity checking is to be performed.

VERTICAL refers to the vertical parity bit on each character transmitted and received. For NSPs/LSPs only, vertical parity generation and testing is suppressed when the character size of the translate table specified by the CODE attribute is 8, or when the NSPs/LSPs are operating in synchronous transparent mode. NSPs/LSPs are not capable of generating and/or checking vertical parity in these instances.

HORIZONTAL refers to the horizontal block check sequence (BCS). If the BCS is a block check character (BCC), then BCCEVEN (initial or final value of all zeros) or BCCODD (initial value of character size ones and final value of all zeros) should be specified; the polynomial in both cases is $X^{<\text{character size}> + 1$. If a nonstandard cyclic redundancy check (CRC) is to be used, the CRC polynomial and initial and final values must be specified.

Table 3-2 shows the values of polynomial, initial polynomial, and final polynomial for the predefined BCS types.

Table 3-2. BCS Type Values for Horizontal Parity Checking

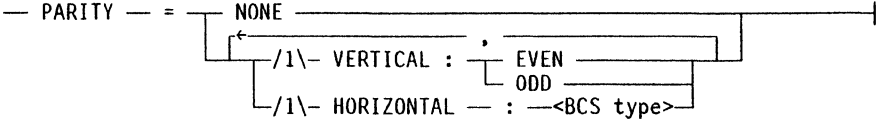
BCS Type	Polynomial	Initial Polynomial	Final Polynomial	Size of Polynomial Field
CRC16	$X^{16}+X^{15}+X^2+1$	0	0	16 bits
CCITTEVEN	$X^{16}+X^{12}+X^5+1$	0	0	16 bits
CCITTODD	$X^{16}+X^{12}+X^5+1$	4"FFFF"	4"1D0F"	16 bits
BCCEVEN	X^8+1	0	0	8 bits
BCCODD	X^8+1	4"FF"	0	8 bits

Notes:

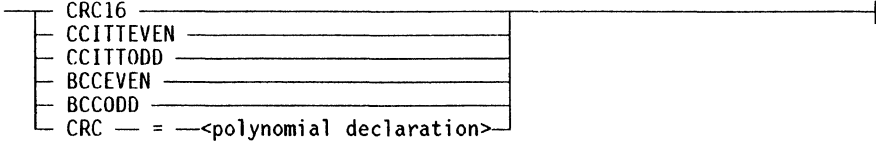
- Normally, BISYNC uses CRC16.
- The hardware-generated FCS for bit mode uses CCITTODD (which is transmitted inverted).
- For BCC-type horizontal parity, the polynomial specified should be of the form $X^n + 1$, where n is the character size of the translate table specified. The initial and final values of BCCEVEN are 0. The initial value of BCCODD is character size ones, and the final value is 0.

Syntax

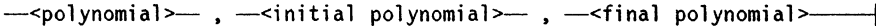
<parity>



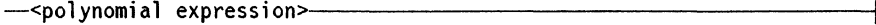
<BCS type>



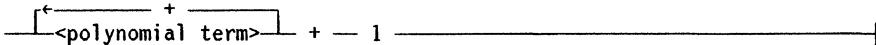
<polynomial declaration>



<polynomial>



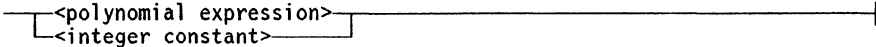
<polynomial expression>



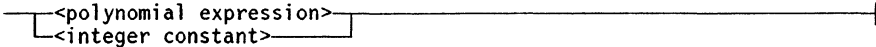
< polynomial term >

Any one of the 16 polynomial terms X1 through X16 (X raised to the first power through X raised to the 16th power).

```
<initial polynomial>
```



<final polynomial>



Example

$$\text{CRC} = X^{16} + X^{12} + X^5 + 1, 0, 0$$

STOPBITS (ASYNC Only)

The STOPBITS attribute specifies the number of stop bits to be transmitted or received at the end of each character. If the character size of the translate table specified by the CODE attribute is 6, 7, or 8, LONG specifies 2 stop bits and SHORT specifies 1 stop bit. If the character size is 5, LONG specifies 1.5 stop bits and SHORT specifies 1 stop bit.

Syntax

<stop bits>

— STOPBITS — = — LONG —
 SHORT

SYNCCONTROL (SYNC Only)

The SYNCCONTROL attribute specifies the characters to be used by the receiver and transmitter to control line synchronization and transparent operation. The constant byte expression defines the character as it appears on the line (the constant byte expression must not be the EBCDIC representation of the character unless the CODE attribute specifies EBCDIC or BINARY). If the character size of the translate table specified by the CODE attribute is less than 8, the character must be right-justified within the constant byte expression with leading zero fill.

Syntax

<sync control>

— SYNCCONTROL — = —————→

→ ————, ————
 /1\ DLE : —<constant byte expression>
 /1\ SYN : —<constant byte expression>

INCLUDE Declaration

The INCLUDE declaration allows variables to be added to the group, line, and station structures in addition to the variables that are predeclared in those structures. Variables declared in the group part are added to one or more group structures, those declared in the line part to one or more line structures, and those declared in the station part to one or more station structures.

Variables declared in an INCLUDE declaration appearing at the protocol module level are added to group, line, and station structures for all groups, lines, and stations in the NDLII program. Variables declared in an INCLUDE declaration in a line control module are added only to the group, line, and station structures for groups, lines, and stations controlled by the algorithm containing the line control module. (Refer to “INCLUDE Declarations in Line Control” in this section.)

Variables declared to be EXTERNAL are accessible to, and can be altered by, the host system through requests to the executive.

The initial value specifications for variables declared in an INCLUDE declaration are overridden if values for those variables are specified in a group INITIALIZE attribute, line INITIALIZE attribute, or station INITIALIZE attribute in the configuration module.

Syntax

<include declaration>

```
— INCLUDE — { /1\-<group part>
               /1\-<line part>
               /1\-<station part> }
```

<group part>

```
— GROUP — ( { <event declaration> ;
               <interlock declaration>
               <variable declaration list> } )
```

<line part>

```
— LINE — <structure declaration part>
```

<structure declaration part>

```
— ( { <event declaration> ;
      <interlock declaration>
      <variable declaration list>
      EXTERNAL — ( <variable declaration list> ) }
```

<station part>

```
— STATION — <structure declaration part>
```

Example

```
INCLUDE LINE(INTERLOCK DATALOCK;
             EXTERNAL(INTEGER PRIORITY:=99;
                      BOOLEAN USEFORMS:=TRUE))
```

SIGNAL and REPLY Declarations (Global)

Information is passed between the host system and the editors and line controls through the signal and reply areas of requests and results. The global signal declaration and global reply declaration are used to describe the format of these areas.

The SIGNAL OUTPUT area and REPLY OUTPUT area of an output request are used by the host system to pass information to the editor and line control defining the method in which the request is to be processed. The REPLY OUTPUT area of an output request is used by the editor and line control to return information about the processing of the output request. If the host system has asked that a status result be returned when the output request has been completed, the executive uses the REPLY OUTPUT area (and other information contained in the output request) to return an OUTPUTSTATUS result.

The REPLY INPUT area of an input result is used by line control and the editor to pass information to the host system about the processing of the input result.

Exactly one SIGNAL OUTPUT, REPLY INPUT, and REPLY OUTPUT declaration must appear in the protocol module.

Syntax

<global signal declaration>

— SIGNAL — OUTPUT — (—<template>—) —————|

<global reply declaration>

— REPLY —

/1*\- INPUT
/1*\- OUTPUT

 (—<template>—) —————|

<template>

—<fixed variable declaration list> —————|
| ; —<template selection> —————|

<fixed variable declaration list>

—<signal-reply declaration list> —————|

<signal-reply declaration list>

—<signal-reply BOOLEAN declaration>—
—<signal-reply byte declaration>—
—<signal-reply enumerated declaration>—
—<signal-reply integer declaration>—
—<signal-reply string declaration>—

 —————|

<template selection>

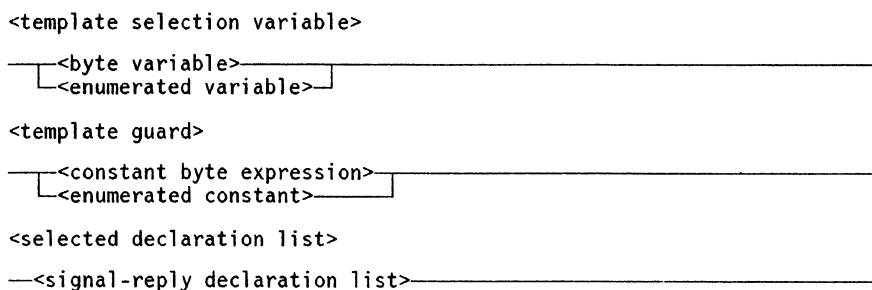
— SELECT —<template selection variable>— (—————→

→

—<template guard>—

 : (—<selected declaration list>—) —————→

→) —————|



Explanation

Variables declared in the template of a global signal declaration or global reply declaration are subject to the following restrictions:

- A template selection variable must be a byte variable or an enumerated variable that is specified within the fixed variable declaration list.
- All template guards must be of the same type (all constant byte expressions or all enumerated constants). In addition, the template guards must be of a compatible type with the template selection variable. If a byte variable is used as a template selection variable, the template guards must be constant byte expressions; if an enumerated variable is the template selection variable, all template guards must be enumerated constants that are specified as values for the enumerated type of the enumerated variable.
- If a variable is declared within the selected declaration list, it must not have been declared in the fixed variable declaration list. A variable that occurs in one selected declaration list can occur in other selected declaration lists but must be of the same type in all cases, and all declarations preceding that variable must be identical in all lists in which the variable appears. <

Except for the previously listed restrictions, identifiers declared in the template of a global signal declaration or a global reply declaration function the same as variables declared within other structures, except that they cannot be given initial values and their initial values are indeterminate.

Variables defined in signal and reply areas are accessed by qualifying the variable name with the name of the structure in which the variable appears. The words SIGNAL and REPLY are not used as qualifiers. For example, STATION.REQUESTXYZ is used in line control to access a variable called XYZ in the SIGNAL OUTPUT area of an output request. Because of this naming convention, the same identifier cannot be declared in both the SIGNAL OUTPUT and REPLY OUTPUT declarations.

The SIGNAL OUTPUT area is Read-Only to editors and line controls; the REPLY INPUT and REPLY OUTPUT areas are Read/Write to editors and line controls.

The information content of message signal and reply areas depends on conventions established between the NDII programmer and the host system.

Example

```
SIGNAL OUTPUT
(BYTE FORMNO;
 STRING DISPLAYSTRING[12];
 SELECT FORMNO(1:(STRING PAGENO[3];
                BOOLEAN LONGFORM);
              2:(STRING PAGENO[3];
                INTEGER NUMLINES);
              3:(BYTE SEPARATOR)
 )
)
```

TERMINALTYPE Declaration

The **TERMINALTYPE** declaration is an **ENUMERATION** declaration that establishes the values for variables of type **TERMINALTYPE**, such as the predeclared station structure variable type.

Example

```
ENUMERATION
  TERMINALTYPE = (TTY,
                  TD830,
                  TC500,
                  SPEEDSENSOR)
```

TRANSLATETABLE Declaration

The TRANSLATETABLE declaration is used to declare translate tables. Translate tables provide a mapping between the EBCDIC character set (used by the data comm subsystem) and the character set used on data comm lines (as required by particular applications). All translate tables have a 256-byte output part and a 256-byte input part. The input portion of a translate table is used by translate statements in editor input process and by the editor and adaptor control TRANSLATEIN function. The output portion of the translate table is used by translate statements in editor output process and by the editor and adaptor control TRANSLATEOUT function.

Syntax

```

<translatetable declaration>
— TRANSLATETABLE —<translatetable ID>—<character size>—————→
→ [ /1*\- INPUT ] <translate body> |
  [ /1*\- OUTPUT ]
<translatetable ID>
—<identifier>—————|
<character size>
— ( [ 5 ] ) —————|
  [ 6 ]
  [ 7 ]
  [ 8 ]
<translate body>
— ( [ <translatetable> , ] [ <translate clause> ] ) —————|
<translatetable>
— [ ASCII ] —————|
  [ BINARY ]
  [ EBCDIC ]
  [ <translatetable ID> ]
<translate clause>
—<constant string expression>— TO [ <constant byte expression> ] |
                                [ <constant string expression> ]

```

Explanation

Translation is performed by indexing the translate table by the binary value of a byte variable and returning the binary value found at that index. The TRANSLATETABLE declaration defines the binary value at each index in the translate table. In the translate clause, each character in the constant string expression to the left of the word TO represents an index into the translate table. If a constant byte expression appears to the right of the word TO, the value of that constant byte expression is the value that is stored at all indexes represented by the constant string expression (a many-to-one mapping). If a constant string expression appears to the right of the word TO, for each index represented by the constant string expression on the left, the character in the corresponding position in the constant string expression on the right is stored (a

one-to-one mapping). In this case, the two constant string expressions must have the same length.

Any index value that does not appear in a translate clause is assigned the value 4"00" (the EBCDIC *NUL* character). If an index value appears more than once, the value assigned is that of the last appearance.

The optional variable <translatetable> indicates that the new translate table is to be initialized to the contents of the specified previously declared or predeclared (ASCII, BINARY, or EBCDIC) translate table. One or more TRANSLATE clauses can then be used to modify the table. A translate table for a line can be changed by the initiate statement in the line control.

BINARY and EBCDIC are synonyms; because the internal code of the data comm subsystem is EBCDIC, specification of BINARY or EBCDIC translation results in translated data that is identical to the untranslated data. An explicit declaration of the ASCII predeclared translate table would be as follows:

```

TRANSLATETABLE SAMEASASCII (7)
  OUTPUT(4"000102030405060708090A0B0C0D0E0F" &
    4"101112131415161718191A1B1C1D1E1F" &
    4"202122232425262728292A2B2C2D2E2F" &
    4"303132333435363738393A3B3C3D3E3F" &
    4"404142434445464748494A4B4C4D4E4F" &
    4"505152535455565758595A5B5C5D5E5F" &
    4"606162636465666768696A6B6C6D6E6F" &
    4"707172737475767778797A7B7C7D7E7F" &
    4"808182838485868788898A8B8C8D8E8F" &
    4"909192939495969798999A9B9C9D9E9F" &
    4"A0A1A2A3A4A5A6A7A8A9AAABACADAFAF" &
    4"B0B1B2B3B4B5B6B7B8B9BABBBCBDBEBF" &
    4"C0C1C2C3C4C5C6C7C8C9CACBCCDCECF" &
    4"D0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF" &
    4"E0E1E2E3E4E5E6E7E8E9EAEBECEDEEEF" &
    4"F0F1F2F3F4F5F6F7F8F9FAFBFCFDFEFF"
    TO
    4"000102030009007F0000000B0C0D0E0F" &
    4"1011121300000800181900001C1D1E1F" &
    4"00000000000A171B000000000050607" &
    4"000016000000004000000001415001A" &
    4"2000000000000000005B2E3C282B21" &
    4"2600000000000000005D242A293B5E" &
    4"2D2F0000000000000007C2C255F3E3F" &
    4"000000000000000000603A2340273D22" &
    4"00616263646566676869000000000000" &
    4"006A6B6C6D6E6F707172000000000000" &
    4"007E737475767778797A000000000000" &
    4"000000000000000000000000000000" &
    4"7B414243444546474849000000000000" &
    4"7D4A4B4C4D4E4F505152000000000000" &
    4"5C00535455565758595A000000000000" &
    4"30313233343536373839000000000000"),
  
```

```

INPUT (4"000102030405060708090A0B0C0D0E0F" &
      4"101112131415161718191A1B1C1D1E1F" &
      4"202122232425262728292A2B2C2D2E2F" &
      4"303132333435363738393A3B3C3D3E3F" &
      4"404142434445464748494A4B4C4D4E4F" &
      4"505152535455565758595A5B5C5D5E5F" &
      4"606162636465666768696A6B6C6D6E6F" &
      4"707172737475767778797A7B7C7D7E7F"
      TO
      4"00010203372D2E2F1605250B0C0D0E0F" &
      4"101112133C3D322618193F271C1D1E1F" &
      4"404F7F7B5B6C507D4D5D5C4E6B604B61" &
      4"F0F1F2F3F4F5F6F7F8F97A5E4C7E6E6F" &
      4"7CC1C2C3C4C5C6C7C8C9D1D2D3D4D5D6" &
      4"D7D8D9E2E3E4E5E6E7E8E94AE05A5F6D" &
      4"79818283848586878889919293949596" &
      4"979899A2A3A4A5A6A7A8A9C06AD0A107");

```

The translate table contains 8-bit characters. The number of bits assembled per character by the receiver and the number of bits per character transmitted by the transmitter are controlled by character size. Character size is 7 for ASCII, 8 for EBCDIC, and 8 for BINARY.

Example

```

TRANSLATETABLE FUNNYASCII (7)
  OUTPUT(ASCII,4"C0D0" TO 4"2B21"),
  INPUT (ASCII,4"11121314" TO CR)

```


Editors

An editor defines an INPUT process and an OUTPUT process that implements application-dependent editing on the text portion of input and output messages according to the requirements of specific terminal types. The INPUT and OUTPUT processes are declared in the NDLII program when an editor body is declared. If EXTERNAL is specified, the code for the editor (including the INPUT and OUTPUT processes) is assumed to be provided externally and is not defined in the NDLII program. Refer to SOURCENDLII for additional information on external editors.

In the configuration module, an editor can be associated with each station. The host system can dynamically change the editor (if any) associated with a station by using a request to the executive. More than one station can be associated with the same editor.

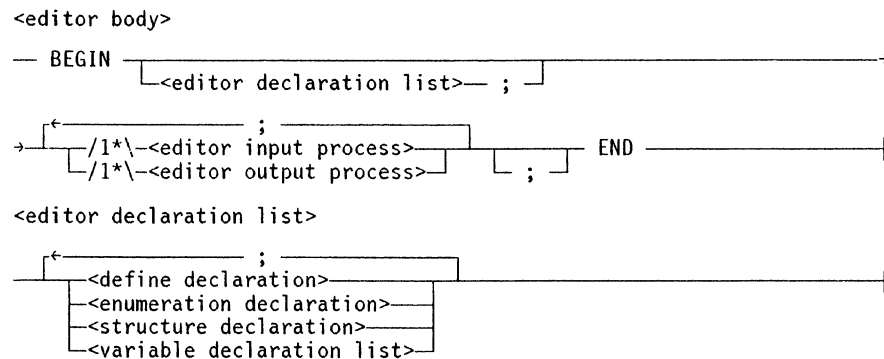
When a station is added to the data comm subsystem as the result of a host request, the executive allocates and initializes the station structure and, if an editor is associated with the station, allocates and initializes the editor global data.

Editor processes do not execute continuously but are invoked by the executive for each message, as required. When a line control process executes a NEXTREQUEST statement to remove from the queue an output message that the host system has placed in the queue for a particular station and an editor is specified for that station, the executive initiates an activation of the editor OUTPUT process and gives this process access to the output message, station structure, and editor global data. The line control process is made to wait until the output editor completes (and is fault-terminated if the editor faults). When the output editor completes, the line control process can transmit the output message and report successful transmission of the message to the host system (if so desired) through execution of a FINISHREQUEST statement.

When an input message arrives for a station for which an editor is specified and a line control process executes a MOVETEXT statement to transfer the text into the RESULT structure of the station, the executive activates the editor INPUT process and gives this process access to the input message, station structure, and editor global data. The line control process is made to wait until the input editor completes (and is fault-terminated if the editor faults). When the input editor completes, the line control process can send the input message to the host system as an input or error result by executing a SENDHOST statement or can discard the message by executing a DISCARDRESULT statement.

Syntax

```
<editor declaration>
— EDITOR —<editor ID> —<editor body>—————|
                        | EXTERNAL |
<editor ID>
—<identifier>—————|
```



< editor input process >

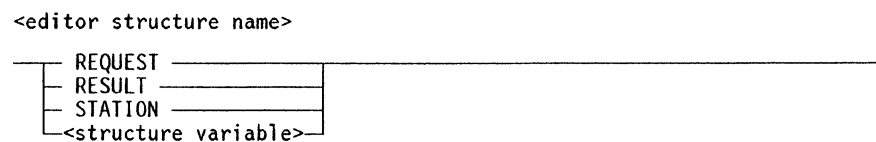
A PROCESS declaration for which the process ID is INPUT.

< editor output process >

A PROCESS declaration for which the process ID is OUTPUT.

Editor Declarations

Syntax



Explanation

An editor INPUT process has access to variables in the RESULT structure for the current message and in the STATION structure for the station from which the message was received. An editor OUTPUT process has access to variables in the REQUEST structure for the current (outgoing) message and in the STATION structure for the station to which the message is routed for transmission. Editor processes are not allowed to access the request currently referenced by STATION.REQUEST or the result currently referenced by STATION.RESULT.

Both processes can declare and access their own structure variables and can access variables in structures declared in the editor declaration list. Variables contained in structures are referenced by qualifying the individual variable identifier with the appropriate editor structure name.

Predeclared Structure Variables in Editors

The REQUEST, RESULT, and STATION structure variables that are predeclared in line control are also predeclared in editors. In addition, the SUPPRESSACTION variable is a RESULT structure variable that is predeclared in editors only.

If SUPPRESSACTION is TRUE when STATION.INPUTACTION is SENDANDWAIT, the line control process that performs a SENDHOST statement for the result is not suspended. SUPPRESSACTION is initially FALSE.

Editor Predeclared Variables

The following variables are predeclared in, and local to, each editor process.

BUFFEROVERFLOW (Boolean, Read-Only)

BUFFEROVERFLOW is assigned TRUE if a TRANSFER statement attempts to append more characters to the DESTINATION variable than the maximum size of DESTINATION allows. BUFFEROVERFLOW is assigned FALSE by a SHORTEN statement that deletes characters from DESTINATION.

DESTINATION (String, Read/Write)

DESTINATION is the string variable returned as the output message of an editor process. DESTINATION can be altered only by the editor ALLOCATE statement, editor TRANSFER statement, editor TRANSLATE statement, and SHORTEN statement. DESTINATION initially has a string maximum size of 0. The ALLOCATE statement must be used to actually obtain space for DESTINATION before any other action can be performed on the variable.

SOURCE (String, Read-Only)

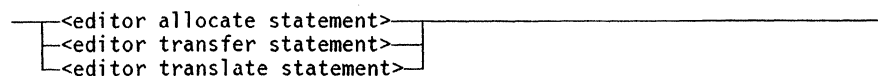
SOURCE is the string variable that represents the input message to an editor process. SOURCE can be changed only by an editor TRANSLATE statement.

Special Editor Statements

The special editor statements that follow can appear only in editor INPUT processes and editor OUTPUT processes. These statements are described in the information that follows. In addition to such statements, the editor processes can include all the general statements.

Syntax

<special editor statement>



Editor ALLOCATE Statement

The editor ALLOCATE statement obtains space for the predeclared string DESTINATION. The maximum size of the DESTINATION string is specified by the

destination maximum size, which can exceed 255 for this special string. If an asterisk (*) or 0 is specified, STATION.MAXINPUT is used if the editor process is the INPUT process, and STATION.MAXOUTPUT is used if the editor process is the OUTPUT process.

Syntax

```
<editor allocate statement>
— ALLOCATE — ( — DESTINATION — , —<destination max size>— ) ———|
<destination max size>
—<integer expression>———|
  | * |
```

Example

```
ALLOCATE(DESTINATION,80)
```

Editor TRANSFER Statement

The editor TRANSFER statement is used to append string expressions to the end of the predeclared string DESTINATION. The repeat part causes a string consisting of repeat count concatenations of the specified string expression to be appended to DESTINATION. If the repeat count is equal to 0, nothing is appended.

If BUFFEROVERFLOW is TRUE, no text is appended. A transfer operation is terminated when one of the following conditions occurs:

- All string expressions have been transferred.
- DESTINATION is already full and more text remains to be appended. BUFFEROVERFLOW is assigned the value TRUE when this condition occurs.

Syntax

```
<editor transfer statement>
— TRANSFER — { —<string expression>— , —<repeat part>— } ———|
<repeat part>
— REPEAT — ( —<string expression>— , —<repeat count>— ) ———|
<repeat count>
—<integer expression>———|
```

Example

```
TRANSFER REPEAT("* ",5), "ERROR--", ERRORSTRING
```

Editor TRANSLATE Statement

The editor TRANSLATE statement replaces the entire contents of the specified string variable by its translation. If the INPUT process of the editor executes an editor TRANSLATE statement, the input part of the translate table is used. If the OUTPUT process executes an editor TRANSLATE statement, the output part of the translate table is used.

Syntax

<editor translate statement>

— TRANSLATE — (—<string variable>— , —<translatetable>—) ———|

Example

```
TRANSLATE(ERRORSTRING,FUNNYASCII)
```

Special Editor Functions

The following functions, which are described in the subsequent text, are valid both in editors and in line controls:

- DIV function
- DROP function
- HEAD function
- INTEGER function
- MOD function
- STRING function
- STRINGADD function
- STRINGSUBTRACT function
- SUBSTRING function
- TAIL function
- TAKE function
- Editor TRANSLATE functions
- TRIM function

DIV Function

The DIV function returns an integer quantity whose value is the quotient resulting from the integer division of the INTEGER expression by the DIV modulus. All integer quantities are treated as 16-bit unsigned values. If the DIV modulus is 0, a DIVIDE BY ZERO error occurs, and the process is fault-terminated.

Syntax

```
<div function>
— DIV — ( —<integer expression>— , —<div modulus>— ) —————|
<div modulus>
—<integer expression>—————|
```

Example

```
I := DIV(INDEX+OFFSET,8)
```

DROP Function

The DROP function returns a string consisting of a copy of the value of the specified string expression after the leftmost drop count characters of the string expression are deleted. If the drop count is greater than the number of characters in the string expression, a STRING PROTECT error occurs, and the process is fault-terminated. If the drop count is 0, a string equal to the value of the string expression is returned. If the drop count equals the length of the string expression, the NULL string is returned.

Syntax

```
<drop function>
— DROP — ( —<string expression>— , —<drop count>— ) —————|
<drop count>
—<integer expression>—————|
```

Example

```
TEMPSTRING := DROP(INSTRING,INDEX-1)
```

HEAD Function

The HEAD function returns a new string consisting of a copy of all contiguous characters at the beginning of the value of the string expression that belong to the set of characters specified by the character set.

If the first character in the string expression is not a member of the character set, the NULL string is returned. If all of the characters in the string expression are in the character set, a string equal to the value of the string expression is returned.

Syntax

<head function>

— HEAD — (—<string expression>— , —<character set>—) —————|

Example

```
SEQSTRING := HEAD(INSTRING,"0123456789")
```

INTEGER Function

The INTEGER function returns the integer value represented by the characters in the string expression. If the string expression does not contain only EBCDIC digits or if the value to be returned is greater than 65535, an INVALID OPERAND error occurs, and the process is fault-terminated.

Syntax

<integer function>

— INTEGER — (—<string expression>—) —————|

Example

```
SEQNUM := INTEGER(SEQSTRING)
```

MOD Function

The MOD function returns an integer quantity whose value is the remainder resulting from the integer division of the INTEGER expression by the MOD modulus. All integer quantities are treated as 16-bit unsigned values. If the MOD modulus is 0, a DIVIDE BY ZERO error occurs, and the process is fault-terminated.

Syntax

<mod function>

— MOD — (—<integer expression>— , —<mod modulus>—) —————|

```

<mod modulus>
—<integer expression>—————|

```

Example

```

BYTESLEFT := MOD(TOTALBYTES,MSGSIZE)

```

STRING Function

The STRING function generates a string that is the EBCDIC representation of the decimal value of the specified integer expression. The RESULT LENGTH parameter specifies the number of characters to be returned as the function result. If an asterisk (*) is specified, the function returns a string with a length exactly sufficient to represent the result. If the RESULT LENGTH is an integer expression, a result of the specified length is returned, with EBCDIC zero ("0") fill in the most significant digits, if necessary. If the specified result length is greater than 255 or if the result is too large to be represented in <result length> characters, a STRING PROTECT error occurs, and the process is fault-terminated.

Syntax

```

<string function>
— STRING — ( —<integer expression>— , —<result length>— ) ———|
<result length>
—<integer expression>—————|
  *

```

Example

```

SEQSTRING := STRING(SEQNUM,6)

```

STRINGADD Function

The STRINGADD function adds two integer quantities that are represented as strings of EBCDIC digits and returns the sum as an integer quantity that is represented as a string of EBCDIC digits. The ADD RESULT LENGTH parameter specifies the number of characters to be returned as the function result. If an asterisk (*) is specified, the function returns a string with a length exactly sufficient to represent the result of the addition. If an integer expression is specified as the add result length, a result of the specified length is returned with EBCDIC zero ("0") fill in the most significant digits, if necessary. If the add result length specified is greater than 255, or if the result is longer than 255 characters and an asterisk was specified, a STRING PROTECT error occurs, and the process is fault-terminated. If the result is too large to be represented in <add result length> characters, the result is truncated at the most significant end.

If either string expression contains a character that is not an EBCDIC digit, an INVALID OPERAND error occurs, and the process is fault-terminated.

Syntax

```

<stringadd function>
— STRINGADD — ( —<string expression>— , —<string expression>— , →
→<add result length>— ) —————|
<add result length>
—<integer expression>—————|
  | *

```

Example

```
TEMPSTRING := STRINGADD(LASTSEQSTR, INCREMENT, 6)
```

STRINGSUBTRACT Function

The STRINGSUBTRACT function performs the tens-complement subtraction (modulo 10 to the <subtract result length> power) of the integer quantity represented by the second string expression from the integer quantity represented by the first string expression. The value of the function is the result of the subtraction represented as a string of EBCDIC digits. The SUBTRACT RESULT LENGTH parameter specifies the number of characters to be returned as the function result. If an asterisk (*) is specified, the function returns a string with a length exactly sufficient to represent the result of the subtraction. If an integer expression is specified as the <subtract result length>, a result of the specified length is returned with EBCDIC zero ("0") fill in the most significant digits, if necessary. If the result is too large to be represented in <subtract result length> characters, it is truncated at the most significant end.

Syntax

```

<stringsubtract function>
— STRINGSUBTRACT — ( —<string expression>— , —<string expression>→
→ , —<subtract result length>— ) —————|
<subtract result length>
—<integer expression>—————|
  | *

```

If either string expression contains a character that is not an EBCDIC digit, an INVALID OPERAND error occurs, and the process is fault-terminated.

Example

```
SEQSTRING := STRINGSUBTRACT(CURRENTSEQ, STEPSIZE, 6)
```

SUBSTRING Function

The SUBSTRING function returns a string that is a copy of the specified part of the value of the string expression. The substring consists of <substring length> contiguous characters copied from the string expression, starting with the character at the position

indicated by <substring start>. If the substring length is 0, the NULL string is returned.

If <substring start> is not in the range 1 to the length of the string expression, or if (<substring start> + <substring length> - 1) is greater than the length of the string expression, a STRING PROTECT error occurs, and the process is fault-terminated.

Syntax

```
<substring function>
— SUBSTRING — ( —<string expression>— , —<substring start>— , —>
→<substring length>— ) —————|
<substring start>
—<integer expression>—————|
<substring length>
—<integer expression>—————|
```

Example

```
TEMPSTRING := SUBSTRING(INFOSTRING, INDEX, FIELDWIDTH)
```

TAIL Function

The TAIL function returns a string consisting of a copy of the value of the specified string expression starting with the first character that does not belong to the set of characters specified by the character set. If the first character in the string expression is not a member of the character set, a string equal to the value of string expression is returned. If all of the characters in the string expression are in the character set, the NULL string is returned.

Syntax

```
<tail function>
— TAIL — ( —<string expression>— , —<character set>— ) —————|
```

Example

```
NEWSTRING := TAIL(TEMPSTRING, " ")
```

TAKE Function

The TAKE function returns a string consisting of a copy of the first <take count> characters of the value of the string expression. If the take count is greater than the number of characters in the string expression, a STRING PROTECT error occurs, and the process is fault-terminated. If the take count is 0, the NULL string is returned. If the take count is equal to the length of the string expression, a copy of the value of the string expression is returned.

Syntax

```
<take function>
— TAKE — ( —<string expression>— , —<take count>— ) —————|
<take count>
—<integer expression>—————|
```

Example

```
TOKEN := TAKE(INSTRING,TOKENLENGTH)
```

Translate Functions

Syntax

TRANSLATEIN and TRANSLATEOUT are byte functions that return the value stored in the specified translate table at the index referenced by the value of the BYTE expression.

Syntax

```
<editor translate function>
┌ TRANSLATEIN ─┐ ( —<translatetable>— , —<byte expression>———→
└ TRANSLATEOUT ┘
→ ) —————|
```

Example

```
LASTCHAR := TRANSLATEOUT(FUNNYASCII,4'13')
```

TRIM Function

The TRIM function returns a string consisting of a copy of the value of the string expression from the first character to the point at which all subsequent characters belong to the set of characters specified by the character set. If the entire string expression consists of characters in the character set, the NULL string is returned. If none of the characters in the string expression are in the character set, a string equal to the value of the string expression is returned.

Syntax

```
<trim function>
— TRIM — ( —<string expression>— , —<character set>— ) —————|
```

Example

```
OUTSTRING := TRIM(OUTSTRING," " & CR)
```

Editor Process Fault Termination

If an editor process commits an irrecoverable error or executes an ABORT statement, the process is terminated. When an INPUT process is terminated, the executive indicates that the editor was terminated in the result returned to the system. When an OUTPUT process is terminated, the executive rejects the request with an indication that the editor terminated abnormally. In addition, the line control process that initiated the editor is fault-terminated.

Algorithms

An algorithm defines a set of line control and adaptor control processes to implement a particular line protocol in conjunction with a particular station management policy. In the configuration module, an algorithm is associated with each line. More than one line can use the same algorithm; one copy of the algorithm is used per line.

In addition to declarations, an algorithm consists entirely of two sections: line control and adaptor control. Because of their importance and uniqueness, line control and adaptor control are discussed in separate subsections of this section. Other aspects of algorithms are described in the following text.

ALGORITHM Declaration

An algorithm consists of declarations, a line control section, and one or more adaptor control sections. The DEFINE declaration and the ENUMERATION declaration are described in Section 2. The DUPLICATE structure declaration, the REPLY declaration, and the SIGNAL declaration are discussed in this section.

From the standpoint of an operating algorithm, only one adaptor control is present. Multiple compatible adaptor control sections can be declared in a single ALGORITHM declaration; however, a particular adaptor control must be selected for each line that uses the ALGORITHM declaration. The adaptor control is selected by means of the line ALGORITHM attribute in the configuration module.

Syntax

```
<algorithm declaration>
— ALGORITHM —<algorithm ID>— BEGIN —<algorithm declaration list>—>

→ ; —<line control>— ; —<adaptor control>— [ ; ] END —————|

<algorithm ID>
—<identifier>—————|

<algorithm declaration list>
[ —<define declaration>— ; —————|
  —<duplicate structure declaration>—|
  —<enumeration declaration>—|
  —/1\—<reply declaration>—|
  —/1\—<signal declaration>—| ]
```

DUPLICATE Structure Declaration

The DUPLICATE structure declaration is used to declare structures that are shared between line control and adaptor control. A local copy of the shared structure is allocated in both line control and adaptor control. The line control COPYSTRUCTURE statement, in conjunction with the adaptor control LOADSTRUCTURE statement, can be used to copy the current information from the line control local copy into the adaptor control local copy. One purpose for transferring such information is to communicate poll lists to adaptor control.

Syntax

```
<duplicate structure declaration>
— DUPLICATE —<structure declaration>—————|
```

Example

```
DUPLICATE STRUCTURE POLL[5] (INTEGER PRIORITY := 0;
                              STRING ADDR[2] := "00")
```

SIGNAL and REPLY Declarations

Information is passed between line control and adaptor control through the signal and reply areas of control blocks. The SIGNAL declaration and REPLY declaration are used to describe the formats of these signal and reply areas.

The signal area of a control block is used by line control to pass information to adaptor control defining the operation to be performed. The reply area of a control block is used by adaptor control to pass information to line control regarding the outcome of that operation. The template is described under “SIGNAL and REPLY Declarations (Global)” earlier in this section.

Variables declared in signal and reply areas are accessed by qualifying the variable name with the name of the structure in which the variable appears. The words SIGNAL and REPLY are not used as qualifiers. For example, a variable called ABC appearing in the signal area of a control block called CB1 is accessed by using the syntax CB1.ABC. Because of this naming convention, the same identifier cannot be declared in both a SIGNAL declaration and a REPLY declaration.

To line control, the signal area is Read/Write and the reply area is Read-Only. To adaptor control, the signal area is Read-Only and the reply area is Read/Write.

Syntax

```
<signal declaration>
— SIGNAL — CB — ( —<template>— ) —————|
<reply declaration>
— REPLY — CB — ( —<template>— ) —————|
```

Line Control

A line control defines one or more processes that control the execution of a particular line protocol by initiating adaptor control processes that implement a particular station management policy.

When a line is added to the data comm subsystem as the result of a host request, the executive does the following:

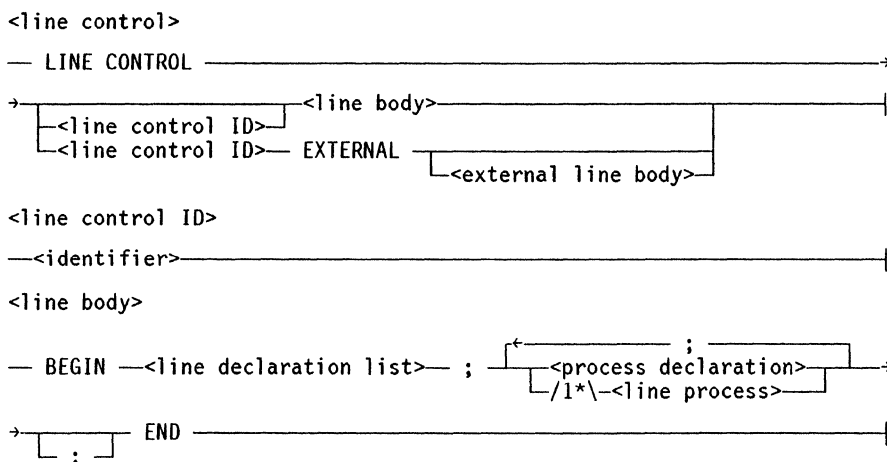
- Allocates and initializes the LINE structure
- Allocates and initializes the line control global data for the algorithm associated with the line (including the line control copy of the duplicate structures)
- Activates the main line control process and gives this process access to the appropriate LINE structure, GROUP structure, and line control global data

The main line control process executes continuously and activates other line control processes by using the PROCESS statement. Each of these activations is given access to the same LINE structure, GROUP structure, and line control global data as are accessed by the main line control process.

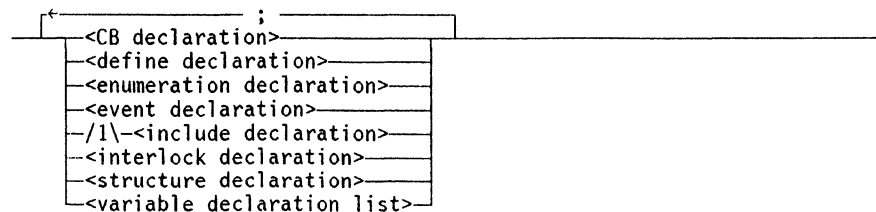
LINE CONTROL Definition

A line control section must include a process called LINE. If a line body is declared, the LINE process appears in the NDLLI program. Additional declarations and processes can be specified as shown in the definitions for line body and line declaration list. When EXTERNAL is specified in place of a line body, the line control code (including the LINE process) is assumed to be provided externally and is not defined in the NDLLI program.

Syntax



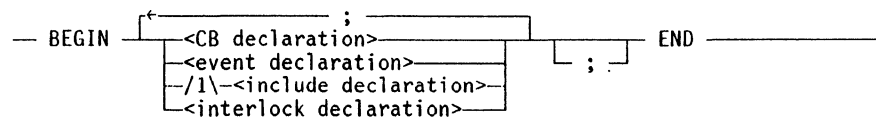
<line declaration list>



<line process>

A PROCESS declaration for which the process ID is LINE.

<external line body>



Explanation

The declarations specified in the external line body must correspond to the declarations required by the external line control. Executing the external line control when the declarations are not complete results in either of two errors. A NON-PRESENT STRUCTURE initialization error results if the required INCLUDE declaration is incorrect. In this case, the data comm subsystem does not initialize. An INVALID ADDRESS error results if any other required declaration is incorrect. In this case, the process is fault-terminated. Refer to SOURCENDLII for additional information on external line controls.

The DEFINE declaration and the ENUMERATION declaration are described in Section 2. Additional line control semantics are given in this section for the INCLUDE declaration, the PROCESS declaration, the STRUCTURE declaration, and the variable declaration list.

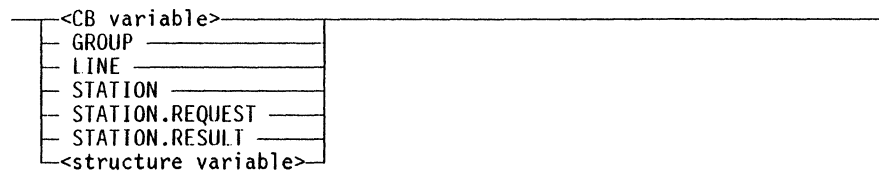
Line Control Declarations

Line control has access to variables in the LINE structure for the line it controls, to variables in the GROUP structure for the group to which the line it controls belongs, to variables in the STATION structures for the stations it controls, and to variables in any CB variables and structure variables declared in line control. Variables in these structures are referenced by qualifying the variable identifier with the appropriate structure name.

Predeclared variables in the CB variable, GROUP, LINE, STATION, STATION.REQUEST, and STATION.RESULT structures are described in the information that follows.

Syntax

<line control structure name>



Predeclared Control Block Structure Variables

The following variables are predeclared in a control block.

ABORTREASON (ABORTTYPE, Read-Only)

ABORTREASON is always NONE unless CATEGORY is ABORTED. If the INITIATE statement specified INPUT or OUTPUT, ABORTREASON contains the abort reason specified in the adaptor control ABORT statement.

CATEGORY (CBCATEGORY, Read-Only)

CATEGORY indicates the status of the operation to which the control block was assigned by an INITIATE statement. The values that CATEGORY can assume are described below:

Value	Description
ABORTED	If the INITIATE statement specified INPUT (or OUTPUT), the operation terminated because the adaptor control INPUT (or OUTPUT) process executed an ABORT statement; the value of ABORTREASON is the abort reason specified in the ABORT statement.
CANCELLED	If the INITIATE statement specified INPUT (or OUTPUT), the operation terminated because of a subsequent INITIATE statement that specified CANCEL INPUT (or OUTPUT), an adaptor control CANCEL INPUT (or CANCEL OUTPUT) statement, or the abnormal termination of a prior input (or output) operation.
COMPLETED	The operation terminated normally. The CATEGORY of a control block that has not yet been initiated is COMPLETED.
INPROGRESS	The operation has been initiated but has not yet terminated.
REJECTED	The operation terminated because one of the preconditions for successful execution was not met. (These preconditions are described under "INITIATE Statement" in this section.)

DCE (Structure, Read-Only)

The DCE substructure provides access to the results returned in a control block initiated by an INITIATE statement that specifies TESTADAPTOR. The values in the substructure are undefined when the DCE substructure is accessed and CB.CATEGORY does not equal COMPLETED or when the previous operation was not an INITIATE statement that specified TESTADAPTOR.

The individual signals are referenced with the syntax *<CB variable>.DCE.<signal name>*, where *<signal name>* is one of the following Boolean variables:

Value	Description
CLEARTOSEND	This signal is generated by the data set in response to the REQUESTTOSEND signal. The CLEARTOSEND signal indicates to the host that the data set is prepared to transmit data.
DATAARRIERDETECT	This signal indicates that a data carrier is being sensed by the data set. When DATAARRIERDETECT is FALSE, it can indicate either of two conditions: the end of the current received message or a fault condition before the end-of-message indicator was received.
DATASETREADY	This signal is generated by the local data set to indicate that the data set is ready to operate. The FALSE condition indicates that the transmission path is not connected.
DATATERMINALREADY	This signal controls switching of the data set to the communications channel. The FALSE condition indicates to the data set that the host is not ready to establish a communications channel.
REQUESTTOSEND	This signal is generated by the host to indicate to the local data set that the host is ready to transmit data. The TRUE condition is maintained when TRANSMITTER.ENABLED is TRUE.
RINGINDICATOR	This signal indicates that a ringing signal is being received from a remote station.
SPECIALINPUT	This signal indicates the current value of RECEIVER.SPECIAL, which is used to control the special input signal on the electrical interface for the line. The field engineer is responsible for strapping this signal to the desired input signal on the line adaptor. This feature is available only on NSPs/LSPs.
SPECIALOUTPUT	This signal indicates the current value of TRANSMITTER.SPECIAL, which is used to control the special output signal on the electrical interface for the line. The field engineer is responsible for strapping this signal to the desired output signal on the line adaptor. This feature is available only on NSPs/LSPs.

TEXT (Reference)

If TEXT is not NULL, it references the buffer used for message communication with adaptor control. TEXT is affected by the ALLOCATE statement, DEALLOCATE statement, MOVETEXT statement, COPYSTRING statement, and COPYSTRUCTURE statement. If TEXT is not NULL, *<CB variable>.TEXT* is a valid string primary that allows line control to examine the contents of the buffer associated with TEXT.

TEXTSIZE (Integer, Read-Only)

TEXTSIZE indicates the number of text characters actually stored in the buffer referenced by TEXT. TEXTSIZE is not necessarily the same as the actual buffer size. TEXTSIZE is 0 if the number of text characters stored was 0 or if TEXT is NULL.

Predeclared GROUP Structure Variables

The following variables in the GROUP structure are predeclared in line control.

LINECOUNT (Integer, Read-Only)

LINECOUNT contains a count of the number of lines in the group. If the line was not declared to be part of a group, LINECOUNT is assigned the value 1.

MAXINPUT (Integer, Read-Only)

MAXINPUT contains the size, in bytes, of the maximum input message that can be received from any station in the group. MAXINPUT is initialized to the largest value of STATION.MAXINPUT for all stations in the group.

Predeclared LINE Structure Variables

The following variables in the LINE structure are predeclared in line control.

ADAPTOREVENT (Event)

ADAPTOREVENT is caused by the executive whenever LINE.ADAPTORREADY or LINE.CONNECTED is assigned the value TRUE.

ADAPTORREADY (Boolean, Read-Only)

ADAPTORREADY is TRUE when a proper communication path exists between adaptor control and line control. If an error is encountered in this path, ADAPTORREADY is assigned the value FALSE, and the host system is sent an error result. If a control block is initiated when ADAPTORREADY is FALSE, the control block is returned with CATEGORY equal to REJECTED.

BUSY (Boolean, Read/Write)

BUSY, in conjunction with LINE.READY, controls the suspension of line control processes by the host. The initial value of BUSY is TRUE until changed by a line control process. If BUSY is FALSE and LINE.READY is assigned FALSE by the host, or if LINE.READY is FALSE and BUSY is assigned FALSE by the programmer, all line control processes are suspended until LINE.READY is assigned the value TRUE by the host. A result message is sent to the host to indicate that the suspension has occurred.

CONNECTED (Boolean, Read-Only)

For nonswitched lines, CONNECTED is always TRUE. For switched lines, CONNECTED is assigned TRUE after a successful dial-in or dial-out sequence has established a connection, and is assigned the value FALSE when the line is disconnected.

DISCONNECTACTION (DISCONNECTPOLICY, Read/Write)

For nonswitched lines, DISCONNECTACTION is ignored. For switched lines, DISCONNECTACTION controls the method used by the executive to connect a disconnected line. If the value of DISCONNECTACTION is changed while LINE.CONNECTED is FALSE, an implicit conditional cancellation of the current action is assumed. DISCONNECTACTION is initialized to the line DISCONNECTACTION attribute and can assume the following values:

Value	Description
AUTOANSWER	This value specifies that the line is attached to a data set that can answer incoming calls. Whenever the data set gives a ring indication, the call is automatically answered.
LINEDIAL	This value specifies that the line is attached to a data set that is not capable of making calls automatically. The connection is made when the outgoing call is manually dialed.

Value	Description
MANUALDIAL	This value specifies that the line is attached to a data set that is not capable of making calls automatically. The connection is made when the outgoing call is manually dialed.
NONE	This value specifies that the line is attached to a data set that can answer incoming calls, although incoming calls are not answered until the MCS performs a <i>SET AUTOANSWER</i> DCWRITE. For more information about the <i>SET AUTOANSWER</i> DCWRITE, refer to the <i>A Series DCALGOL Programming Reference Manual</i> .
STNSETDIAL	This value specifies that the line is attached to a data set that has an automatic calling unit (ACU) for originating outgoing calls.
STNSETDIALANSWER	This value specifies that the line is attached to a data set that has an ACU for originating outgoing calls. Unlike <i>LINEDIAL</i> , a ring indication causes the call to be automatically answered.

READY (Boolean, Read-Only)

READY, in conjunction with LINE.BUSY, controls the suspension of line control processes by the host. If READY is FALSE and LINE.BUSY is assigned the value FALSE by the programmer, or if LINE.BUSY is FALSE and READY is assigned the value FALSE by the host, all line control processes are suspended until READY is assigned the value TRUE by the host. A result message is sent to the host to indicate that the suspension has occurred. READY is initialized to the value of the line READY attribute.

STATUSCHANGED (Boolean, Read/Write)

STATUSCHANGED is assigned the value TRUE when the status of the line or any station associated with the line changes significantly. If the line is associated with a stationset, LINE.STATUSCHANGED responds to station changes only when the station is a member of the associated stationset. If stationsets have been declared for the line but the line is currently not associated with a stationset, LINE.STATUSCHANGED responds to station changes only for stations in unassociated stationsets.

LINE.STATUSCHANGED is assigned the value TRUE when one of the following conditions occurs (whether or not the affected line or station is READY):

- A line or station variable declared as EXTERNAL in an INCLUDE declaration is changed as the result of a host request.
- A predeclared line or station variable is changed as the result of a host request.
- A station is added, cleared, or deleted.

The LINE.SYSTEM event is caused by the executive whenever LINE.STATUSCHANGED is assigned the value TRUE.

SYSTEM (Event)

The SYSTEM event is caused by the executive whenever LINE.STATUSCHANGED is assigned the value TRUE or when STATION.QUEUED changes from FALSE to TRUE for any station associated with the line. If the line is associated with a stationset, LINE.SYSTEM responds to STATION.QUEUED only when the station is a member of the associated stationset. If stationsets have been declared for the line but the line is currently not associated with a stationset, LINE.SYSTEM responds to STATION.QUEUED only for stations in unassociated stationsets.

Predeclared STATION Structure Variables

The following variables in the STATION structure are predeclared in line control.

CONTROL (Byte, Read-Only)

CONTROL contains the control character of the station and is initialized to the value of the STATION CONTROL attribute.

ENABLED (Boolean, Read-Only)

ENABLED is initialized to the value of the STATION ENABLED attribute and is modified only by request of the host. If ENABLED is TRUE, the host has indicated that it is prepared to accept input from this station. If ENABLED is FALSE, line control should not return any input messages from the station, although the executive does not prevent these messages from being returned.

INPUTACTION (INPUTPOLICY, Read-Only)

INPUTACTION controls the action that the executive performs on input results placed in a queue for the host by the SENDHOST statement. INPUTACTION is initialized to the value of the STATION INPUTACTION attribute and can assume the following values:

Value	Description
SEND	The result is sent to the host.
SENDANDWAIT	The predeclared result variable SUPPRESSACTION is interrogated. If the value of this variable is FALSE, the result is sent to the host and the process that performed the SENDHOST statement is suspended until the result is acknowledged by the host. If the value of this variable is TRUE, the result is sent to the host and the process that performed the SENDHOST statement is allowed to continue.

LINEWIDTH (Integer, Read-Only)

LINEWIDTH contains the number of characters per terminal line and is initialized to the value of the TERMINAL LINEWIDTH attribute.

MAXINPUT (Integer, Read-Only)

MAXINPUT contains the size, in bytes, of the maximum input message from the station and is initialized to the value of the TERMINAL MAXINPUT attribute.

MAXOUTPUT (Integer, Read-Only)

MAXOUTPUT contains the size, in bytes, of the maximum output message that can be sent to the station as one transmission and is initialized to the value of the TERMINAL MAXOUTPUT attribute.

PAGECOUNT (Integer, Read-Only)

PAGECOUNT contains the number of pages the terminal can buffer and is initialized to the value of the TERMINAL PAGECOUNT attribute.

PAGESIZE (Integer, Read-Only)

PAGESIZE contains the number of lines per terminal page and is initialized to the value of the TERMINAL PAGESIZE attribute.

QUEUED (Boolean, Read-Only)

QUEUED is TRUE if the queue of output requests for the station is not empty. If the queue is empty and the executive receives an output request for the station, the executive causes the LINE.SYSTEM event for the lines for which the station is visible (in the sense of the NEXTSTATION statement).

READY (Boolean, Read-Only)

If READY is TRUE, a line control process can reference the station by executing a NEXTSTATION statement to assign to its station reference variable the value for that station. READY is initialized to the value of the STATION READY attribute and can be modified by the host system (through a request to the executive) or assigned the value FALSE by a line control process through execution of a SENDHOST ERROR statement or a CLEAR station statement.

RECEIVEADDRESS (String, Read-Only)

RECEIVEADDRESS contains the receive address characters for the station and is initialized to the value of the RECEIVE portion of the STATION ADDRESS attribute.

REQUEST (Reference)

If REQUEST is not NULL, it references an output request that is treated as a structure. In addition to the predeclared structure variables listed below, the variables declared as part of the SIGNAL OUTPUT and REPLY OUTPUT declarations are part of this structure. If REQUEST is NULL, an attempt to reference one of the structure variables causes an INVALID OPERAND error and results in fault-termination of the referencing process. REQUEST is affected by the DEQUEUE statement, ENQUEUE statement, FINISHREQUEST statement, and NEXTREQUEST statement.

The following three variables are predeclared structure variables of REQUEST:

Variable	Description
RESULTREQUIRED (Boolean, Read-Only)	RESULTREQUIRED is TRUE if the host system requires a status result to be returned when a request has been completed.
TEXT (Reference)	If TEXT is not NULL, it references the message to be sent to the remote station. TEXT is affected by the MOVETEXT statement. If TEXT is not NULL, STATION.REQUEST.TEXT is a valid string primary; thus, line control is able to examine the contents of the buffer associated with TEXT.
TEXTSIZE (Integer, Read-Only)	TEXTSIZE indicates the number of text characters stored in the buffer referenced by TEXT. (TEXTSIZE is not necessarily the same as the buffer size.) TEXTSIZE is 0 if no text characters are stored or if TEXT is NULL. Statements that affect TEXT correspondingly affect TEXTSIZE.

REQUESTLIST (List)

REQUESTLIST is a list in which output requests can be stored (in association with keys) for later retrieval. REQUESTLIST is affected by the DEQUEUE statement, ENQUEUE statement, and PURGE statement.

RESULT (Reference)

If RESULT is not NULL, it references an input result that is treated as a structure. In addition to the predeclared structure variables listed below, the variables declared as part of the REPLY INPUT declaration are part of this structure. If RESULT is NULL, an attempt to reference one of the structure variables causes an INVALID OPERAND error and results in fault-termination of the referencing process. RESULT is affected by the DEQUEUE statement, ENQUEUE statement, NEWRESULT statement, and SENDHOST statement.

The following three variables are predeclared structure variables of RESULT:

Variable	Description
CONTROLMESSAGE (Boolean, Read/Write)	If CONTROLMESSAGE is TRUE, the result is marked as a control message if it is returned to the host system as an input result. Conventionally, CONTROLMESSAGE is assigned the value TRUE if the first character of TEXT matches STATION.CONTROL.
TEXT (Reference)	If TEXT is not NULL, it references the message received from the remote station. TEXT is affected by the MOVETEXT statement. If TEXT is not NULL, STATION.RESULT.TEXT is a valid string primary; thus, line control is able to examine the contents of the buffer associated with TEXT.
TEXTSIZE (Integer, Read-Only)	TEXTSIZE indicates the number of text characters stored in the buffer referenced by TEXT. (TEXTSIZE is not necessarily the same as the buffer size.) TEXTSIZE is 0 if no text characters are stored or if TEXT is NULL. Statements that affect TEXT correspondingly affect TEXTSIZE.

RESULTLIST (List)

RESULTLIST is a list in which input results can be stored (in association with keys) for later retrieval. RESULTLIST is affected by the DEQUEUE statement, ENQUEUE statement, and PURGE statement.

SCREEN (Boolean, Read-Only)

If SCREEN is TRUE, the terminal associated with the station is a screen device. SCREEN is initialized to the value of the TERMINAL SCREEN attribute.

TRANSMITADDRESS (String, Read-Only)

TRANSMITADDRESS contains the transmit address characters for the station and is initialized to the value of the TRANSMIT part of the STATION ADDRESS attribute.

TYPE (TERMINALTYPE, Read-Only)

TYPE is initialized to the value of the TERMINAL TYPE attribute, which can assume values declared as enumerated constants in the TERMINALTYPE declaration.

USAGECOUNT (Integer, Read-Only)

USAGECOUNT contains a count of the station references and editor processes that are currently referencing the STATION structure.

WRAPAROUND (Boolean, Read-Only)

WRAPAROUND is TRUE if the terminal performs a carriage return and line feed when it has received STATION.LINEWIDTH or more characters. WRAPAROUND is initialized to the value of the TERMINAL WRAPAROUND attribute.

Line Control Predeclared Variables

The station reference variables (described under “Station Reference Variables” in this section) and the variables that follow are predeclared in line control.

NOMEMORY (Boolean, Read-Only)

NOMEMORY is assigned the value TRUE when the executive detects a shortage of allocatable memory for NDLII processes. It is assigned the value FALSE when an adequate amount of allocatable memory exists.

SYSTEMWAIT (Boolean, Read-Only)

SYSTEMWAIT is assigned the value TRUE in response to a host system wait request and is assigned the value FALSE by a host system resume request.

TIMESTAMP (String[8], Read-Only)

TIMESTAMP is a string variable composed of eight integers that represent hours, minutes, seconds, hundredths-of-a-second, day-of-year DIV 100, day-of-year modulo 100, year DIV 100, and year modulo 100 in string locations 1 through 8. The range of values for each of these integers is specified as follows:

Time Category	Value Range
Hours	0 through 23
Minutes	0 through 59
Seconds	0 through 59
Hundredths-of-a-second	0 through 99
Day-of-year (DIV 100)	0 through 03
Day-of-year (MOD 100)	0 through 99
Year (DIV 100)	19 through 20
Year (MOD 100)	0 through 99

Day-of-year is two consecutive bytes (an integer) in the range 1 through 366; year is also two consecutive bytes (an integer) in the range 1900 through 2099.

Station Reference Variables

The stations that are assigned to a line are declared in a line station list in the configuration module. Each of these stations has an associated STATION structure, which contains predeclared and (optionally) user-declared variables that describe the characteristics and current state of the particular station. These variables are accessed, in the typical fashion for variables in structures, by qualifying the variable name with the structure name (STATION).

Because many STATION structures can be present (one for each station in the station list), each process in line control has a predeclared station reference variable, which is used to select a particular STATION structure as the structure that is referenced when the qualifier *STATION* is used. When the line control LINE process is initiated, the corresponding station reference variable is assigned the value NULL, to indicate that no STATION structure is currently referenced. Other processes inherit the station reference variable of their parent process at the time they are initiated.

If *STATION* is used as a qualifier when the station reference variable is NULL, an INVALID OPERAND error occurs, and the process is fault-terminated. The NULLREFERENCE function can be used to determine whether or not the station reference variable is NULL.

The station reference variable is assigned to reference a particular STATION structure by the NEXTSTATION statement. This statement allows the programmer to select either the next station in the station list or the next station in the list that meets the selection criterion specified in the NEXTSTATION statement. Stations that are not ready (stations for which STATION.READY is FALSE) are not eligible to be selected. The station reference variable is assigned the value NULL either by the NULLSTATION statement or by the NEXTSTATION statement (under certain conditions specified in the discussion of the NEXTSTATION statement).

Inconsistent operation could occur if the executive were to clear or delete a station while a station reference variable was referencing that STATION structure. For this reason, the executive does not clear or delete a station until there are no references to that STATION structure. To allow for maximum flexibility and minimum delay, the station reference variable should be assigned the value NULL whenever it is not being used.

Example

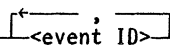
```
CONTROL BLOCK
  INCB[2],
  CTRL
```

EVENT Declaration

An EVENT declaration is used to declare event variables. These events can then be caused using the CAUSE statement and waited on using the WAIT statement and the WAIT function. The state of an event can be interrogated using the HAPPENED function. The initial state of all events is NOT HAPPENED.

Syntax

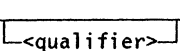
<event declaration>

— EVENT —  —

<event ID>

— <identifier> —

<event variable>

 <event ID> —

Example

```
EVENT
  SYNCEVENT,
  FINISHED
```

INCLUDE Declarations in Line Control

Variables can be added to GROUP, LINE, and STATION structures using the syntax for the INCLUDE declaration. (Refer to “INCLUDE Declaration” in this section.) However, variables added using an INCLUDE declaration appearing in line control are added to the GROUP, LINE, and STATION structures only for groups, lines, and stations controlled by the algorithm associated with that line control module. These variables are included in the GROUP, LINE, and STATION structures in addition to predeclared variables and in addition to variables added by any global INCLUDE declarations (that is, INCLUDE declarations appearing at the protocol module level).

INTERLOCK Declaration

The INTERLOCK declaration is used to declare locks. Locks allow control of access to variables that are shared between processes.

Syntax

<interlock declaration>

```

— INTERLOCK —┐┌<lock ID>┐┐
<lock ID>
—<identifier>—
<lock variable>
┐┌<qualifier>┐┐<lock ID>—

```

Example

```

INTERLOCK
  DATALOCK,
  TABLELOCK

```


PROCESS Declarations in Line Control

Line control is the only environment in which optional processes can be declared. Line control processes are declared using the syntax described for the general PROCESS declaration. One process must be declared with LINE as its process ID. The executive automatically initiates the LINE process after the host system has initialized the line; when the LINE process terminates (normally or otherwise), the host system must reinitiate the line.

Optional processes are initiated by the PROCESS statement. Only the LINE process can initiate other processes. Each process runs asynchronously with all other processes.

STRUCTURE Declarations in Line Control

Structures can be declared in line control using the syntax described for the general STRUCTURE declaration. Unlike duplicate structures declared in algorithms (globally), space for a structure declared in line control is allocated only in the physical environment of line control. These structures cannot be communicated to adaptor control. Care should be taken when accessing variables if a structure is shared between processes, because no automatic interlocking mechanism is provided.

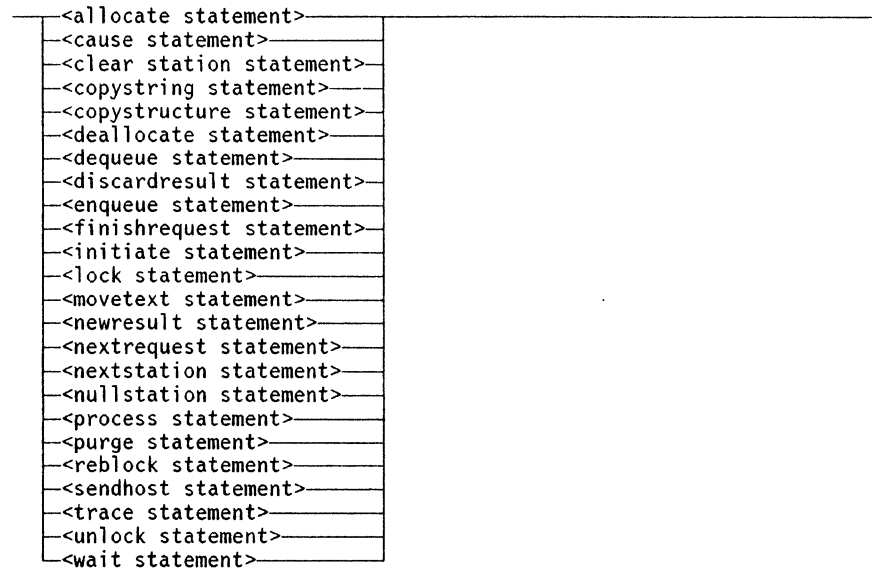
Variable Declarations in Line Control

Boolean variables, byte variables, enumerated variables, integer variables, and string variables can be declared in line control using the syntax described for the general variable declarations. However, because multiple processes can be declared in line control, care should be taken when accessing these variables, because no automatic interlocking mechanism is provided.

Special Line Control Statements

In addition to the general statements described in Section 2, line control processes can include the special line statements shown in the following syntax diagram. Each statement is discussed in detail in the text that follows.

<special line statement>



ALLOCATE Statement

The ALLOCATE statement obtains a buffer, associates it with <CB variable>.TEXT, and assigns 0 to <CB variable>.TEXTSIZE. The length of the buffer, in bytes, is specified by the buffer size variable. If an asterisk (*) or 0 is specified as the buffer size, GROUPMAXINPUT is used as the length of the buffer. If <CB variable>.CATEGORY is INPROGRESS, an INVALID OPERAND error occurs; if <CB variable>.TEXT is not NULL, an OCCUPIED DESTINATION error occurs. In either case, the process is fault-terminated.

Syntax

<allocate statement>

— ALLOCATE — (—<CB variable>— , —<buffer size>—) —————|

<buffer size>

—<integer expression>—————|
 * —————|

Examples

```
ALLOCATE(CB[1],*)
```

```
ALLOCATE(INPUTCB,80)
```

CAUSE STATEMENT

The CAUSE statement changes the state of the event variable to HAPPENED, which allows processes waiting on the event variable to resume execution.

Syntax

<cause statement>

— CAUSE —<event variable>—————|

Example

```
CAUSE SYNCEVENT
```

CLEAR STATION Statement

The CLEAR STATION statement temporarily suspends all activity for the station referenced by the station reference variable. If the station reference variable is NULL, a NULL STATION error occurs, and the process is fault-terminated.

Syntax

<clear station statement>

— CLEAR STATION —————|

Explanation

When a CLEAR STATION statement is executed, STATION.READY is assigned the value FALSE and LINE.STATUSCHANGED is assigned the value TRUE for all lines that have visibility of the station. No further action is performed until STATION.USAGECOUNT becomes 0.

When STATION.USAGECOUNT becomes 0 for a station that is to be cleared, the executive purges STATION.REQUEST, STATION.RESULT, STATION.REQUESTLIST, STATION.RESULTLIST, and the station output queue. A result message is then sent to the host system indicating that the station was cleared.

COPYSTRING Statement

The COPYSTRING statement does the following:

1. Obtains a buffer large enough to contain a copy of the value of the specified string expression
2. Associates this buffer with <CB variable>.TEXT
3. Copies the value of the string expression into the buffer
4. Assigns to <CB variable>.TEXTSIZE the value LENGTH(<string expression>)

If <CB variable>.CATEGORY is INPROGRESS, an INVALID OPERAND error occurs; if <CB variable>.TEXT is not NULL, an OCCUPIED DESTINATION error occurs. In either case, the process is fault-terminated.

Syntax

<copystring statement>

— COPYSTRING — (—<string expression>— , —<CB variable>—) ———|

COPYSTRUCTURE Statement

The COPYSTRUCTURE statement does the following:

1. Obtains a buffer large enough to contain a copy of the source structure
2. Associates this buffer with <CB variable>.TEXT
3. Copies the source structure into the buffer
4. Assigns to <CB variable>.TEXTSIZE the size of the source structure

The source structure must have been declared in a DUPLICATE structure declaration in the ALGORITHM declaration list for the algorithm in which this line control section appears. If <CB variable>.CATEGORY is INPROGRESS, an INVALID OPERAND error occurs; if <CB variable>.TEXT is not NULL, an OCCUPIED DESTINATION error occurs. In either case, the process is fault-terminated.

Syntax

```
<copystructure statement>
— COPYSTRUCTURE — ( —<source structure>— , —<CB variable>— ) —|
<source structure>
—<structure ID>—|
  |<structure array ID>|
```

Example

```
COPYSTRUCTURE (STRUCTUREV,OUTPUTCB)
```

DEALLOCATE Statement

The DEALLOCATE statement does the following:

1. Releases the buffer associated with <CB variable>.TEXT
2. Assigns the value NULL to <CB variable>.TEXT
3. Assigns 0 to <CB variable>.TEXTSIZE

If <CB variable>.CATEGORY is INPROGRESS, an INVALID OPERAND error occurs; if <CB variable>.TEXT is NULL, an EMPTY SOURCE error occurs. In either case, the process is fault-terminated.

Syntax

```
<deallocate statement>
— DEALLOCATE —<CB variable>—|
```

Example

```
DEALLOCATE INCB[1]
```

DEQUEUE Statement

If REQUEST is specified, the DEQUEUE statement removes the request from the STATION.REQUESTLIST queue that is associated with the designated key variable and associates this request with STATION.REQUEST. If a request with the designated key variable does not exist in STATION.REQUESTLIST, STATION.REQUEST is made equal to NULL. The following error conditions result in fault-termination of the process:

- The station reference variable is NULL (NULL STATION error).
- STATION.REQUEST is not NULL (OCCUPIED DESTINATION error).

If RESULT is specified, the DEQUEUE statement removes from the queue the result in STATION.RESULTLIST that is associated with the designated key variable and associates this result with STATION.RESULT. If a result with the designated key variable does not exist in STATION.RESULTLIST, STATION.RESULT is made equal to NULL. The following error conditions result in fault-termination of the process:

- The station reference variable is NULL (NULL STATION error).
- STATION.RESULT is not NULL (OCCUPIED DESTINATION error).

Syntax

<dequeue statement>

— DEQUEUE — (

REQUEST
RESULT

 , —<key>—) —————|

<key>

—<integer expression>—————|

Examples

```
DEQUEUE (REQUEST,3)
```

```
DEQUEUE (RESULT,STATION.INTV)
```

DISCARDRESULT Statement

The DISCARDRESULT statement discards the message referenced by STATION.RESULT. If the station reference variable is NULL, a NULL STATION error occurs; if STATION.RESULT is NULL, an EMPTY SOURCE error occurs. In either case, the process is fault-terminated.

Syntax

<discardresult statement>

— DISCARDRESULT —————|

ENQUEUE Statement

If REQUEST is specified, the ENQUEUE statement takes the request associated with STATION.REQUEST, places it in the STATION.REQUESTLIST queue (associating it with the specified key), and then assigns NULL to STATION.REQUEST. The following error conditions result in fault-termination of the process:

- The station reference variable is NULL (NULL STATION error).
- STATION.REQUEST is NULL (EMPTY SOURCE error).
- A request in STATION.REQUESTLIST is already associated with the specified key (OCCUPIED DESTINATION error).
- The key exceeds 255 (INVALID OPERAND error).

If RESULT is specified, the ENQUEUE statement takes the result associated with STATION.RESULT, places it in the STATION.RESULTLIST queue (associating it with the specified key), and then assigns NULL to STATION.RESULT. The following error conditions result in fault-termination of the process:

- The station reference variable is NULL (NULL STATION error).
- STATION.RESULT is NULL (EMPTY SOURCE error).
- A result in STATION.RESULTLIST is already associated with the specified key (OCCUPIED DESTINATION error).
- The key exceeds 255 (INVALID OPERAND error).

Syntax

<enqueue statement>

— ENQUEUE — (REQUEST
RESULT , —<key>—) —————|

<key>

—<integer expression>—————|

Examples

ENQUEUE (REQUEST, 3)

ENQUEUE (RESULT, STATION.INTV)

FINISHREQUEST Statement

The FINISHREQUEST statement generates an OUTPUTSTATUS result, if required by the host, and assigns NULL to STATION.REQUEST. If either the station reference variable or STATION.REQUEST is NULL, an EMPTY SOURCE error occurs, and the process is fault-terminated.

Syntax

<finishrequest statement>

— FINISHREQUEST —————|

INITIATE Statement

The INITIATE statement starts an asynchronous operation with the specified <CB variable> assigned for communication and synchronization with the operation. If <CB variable>.CATEGORY is INPROGRESS or if <CB variable>.TEXT is not NULL when a special operation is initiated, an INVALID OPERAND error occurs, and the line control process is fault-terminated.

The INITIATE statement changes the state of the implicit event associated with the <CB variable> to NOT HAPPENED, changes the value of the predeclared control block variable CATEGORY to INPROGRESS, and notifies the executive of the operation to be performed. The line control process continues execution without waiting for the operation to terminate. (The WAIT statement or WAIT function can be used to wait for operation termination.)

Syntax

<initiate statement>

— INITIATE — (—<CB variable>— , —<operation type>—) —————|

<operation type>

—<adaptor control operation>—————|
 |<special operation>—————|

<adaptor control operation>

— INPUT —————|
 | OUTPUT —————|
 | CANCEL —————| INPUT —————|
 | | OUTPUT —————|

<special operation>

— DISCONNECT —————|
 | LOAD —————|<class ID>—————|
 | |<translatetable>—————|
 | TESTADAPTOR —————|

Explanation

The following general considerations apply to all operation types. When an operation completes, the executive updates <CB variable> to reflect the status of the operation and then causes the implicit event associated with <CB variable>. Unless the

operation is a TESTADAPTOR operation, any of the following conditions cause the operation to terminate with `<CB variable>.CATEGORY` equal to REJECTED:

- `LINE.ADAPTORREADY` is FALSE.
- `LINE.CONNECTED` is FALSE, and `LINE.DISCONNECTACTION` is not NONE.
- The operation is a special operation, and another operation is still in progress.
- The operation is an adaptor control operation, and a special operation is still in progress.

The TESTADAPTOR operation is accepted at any time.

The individual operation types are described as follows:

Operation Type	Description
INPUT	An input operation executes the adaptor control INPUT process after the successful completion of any prior input operation. Before the start of the operation, <code><CB variable>.TEXTSIZE</code> is assigned the value 0. When the operation terminates, <code>TEXTSIZE</code> is updated to reflect the actual number of TEXT bytes, if any, stored in the buffer by adaptor control. When any prior input operation completes successfully, the adaptor control INPUT process control block structure is initialized from the <code><CB variable></code> and the INPUT process is invoked; otherwise, the operation terminates with <code><CB variable>.CATEGORY</code> assigned CANCELLED, and the INPUT process is not invoked. The operation terminates with <code><CB variable>.CATEGORY</code> assigned the value REJECTED if more than one other input operation is still in progress.
OUTPUT	An output operation executes the adaptor control OUTPUT process after the successful completion of any prior output operation. The number of TEXT bytes, if any, available to adaptor control for the operation is controlled by <code><CB variable>.TEXTSIZE</code>. When any prior output operation completes, the adaptor control OUTPUT process control block structure is initialized from the <code><CB variable></code>, and the OUTPUT process is invoked; otherwise, the operation terminates with <code><CB variable>.CATEGORY</code> assigned CANCELLED, and the OUTPUT process is not invoked. The operation terminates with <code><CB variable>.CATEGORY</code> assigned the value REJECTED if more than one other output operation is still in progress.
CANCEL	A cancel operation conditionally cancels all input or output operations still in progress. The cancel operation terminates with <code><CB variable>.CATEGORY</code> assigned the value REJECTED if no operations of the specified type are in progress.

Operation Type	Description
DISCONNECT	An input or output operation that is waiting for the successful completion of a prior input or output operation is unconditionally canceled. An input or output operation that has already invoked the adaptor control INPUT or OUTPUT process is notified of the cancellation. If the adaptor control process NOCANCEL variable is FALSE, the process is terminated and the operation canceled; otherwise, the cancellation is suspended until either NOCANCEL is assigned the value FALSE (in which case the process is terminated and the operation canceled), or the process terminates for some unrelated reason (in which case the cancel is ignored). A cancel operation can leave the transceiver in an unknown state; for this reason, good practice requires testing the receiver or transmitter to see if it is enabled and, if so, disabling it. The receiver or transmitter can then be enabled and the reception or transmission begun. A disconnect operation breaks a connection on a switched line. If LINE.CONNECTED is FALSE or if the line is not switched, the operation is terminated with <CB variable>.CATEGORY assigned the value REJECTED. If the disconnect attempt is successful, LINE.CONNECTED is assigned the value FALSE.
LOAD	A load operation replaces either the current class information or the current translate table in adaptor control, depending on whether a class ID or a translate table is specified in the INITIATE statement. This operation obtains a buffer, copies the class information or translate table, and associates the buffer with the <CB variable>.TEXT. A deallocate must be performed to release the buffer before another allocate can be performed. If a class ID is specified, the mode of the specified line class must be compatible with the type of the algorithm. An operation of this type allows the necessary changes in line parameters to be made for speedsensor lines. Note <i>Some RECEIVER and TRANSMITTER structure variables in adaptor control are reinitialized when line class information is changed.</i> If a translate table is specified and the character size of that translate table differs from the current character size, and if the line is using horizontal parity, the degree of the BCS polynomial cannot be compatible.
TESTADAPTOR	A TESTADAPTOR operation determines the logic state of DCE signals for the specified line. The status information is returned in the predeclared control block structure variable DCE.

Examples

```
INITIATE(MYCB,TESTADAPTOR)

INITIATE(CBOUT,OUTPUT)

INITIATE(INCB[2],INPUT)

INITIATE(CTRL,DISCONNECT)

INITIATE(MYCB,LOAD DABCLASS)

INITIATE(MYCB,LOAD SPECIALTRANSTABLE)
```

LOCK Statement

The LOCK statement causes the specified lock to be obtained (locked). If the lock is already locked when the LOCK statement is executed, the process is suspended until the lock becomes available. In all circumstances, the process has obtained the lock before the statement following the LOCK statement is executed.

The specified lock must be released (unlocked) before the process terminates. If the process attempts to terminate without unlocking the lock, or if the process fault-terminates for an unrelated reason, the executive unlocks the lock.

Syntax

<lock statement>

— LOCK —<lock variable>—————|

Example

```
LOCK LINE.TABLELOCK
```

MOVETEXT Statement

The <request-to-CB> form of the MOVETEXT statement takes the buffer associated with STATION.REQUEST.TEXT, associates this buffer with <CB variable>.TEXT, and then assigns NULL to STATION.REQUEST.TEXT. The following error conditions result in fault-termination of the process:

- The station reference variable is NULL (NULL STATION error).
- STATION.REQUEST is NULL, or <CB variable>.CATEGORY is INPROGRESS (INVALID OPERAND error).
- STATION.REQUEST.TEXT is NULL (EMPTY SOURCE error).
- <CB variable>.TEXT is not NULL (OCCUPIED DESTINATION error).

The <CB-to-request> form of the MOVETEXT statement takes the buffer associated with <CB variable>.TEXT, associates this buffer with STATION.REQUEST.TEXT, and then assigns NULL to <CB variable>.TEXT. The following error conditions result in fault-termination of the process.

- The station reference variable is NULL (NULL STATION error).
- STATION.REQUEST is NULL, or <CB variable>.CATEGORY is INPROGRESS (INVALID OPERAND error).
- <CB variable>.TEXT is NULL (EMPTY SOURCE error).
- STATION.REQUEST.TEXT is not NULL (OCCUPIED DESTINATION error).

The <CB-to-result> form of the MOVETEXT statement takes the buffer associated with <CB variable>.TEXT, associates this buffer with STATION.RESULT.TEXT, and then assigns NULL to <CB variable>.TEXT. If an editor is specified for the station, the executive then initiates an activation of the editor input process and gives this process access to the input message, STATION structure, and editor global data. The line control process is made to wait until the editor input process completes, at which time the edited input text is contained in STATION.RESULT.TEXT. The following error conditions result in fault-termination of the process:

- STATION.RESULT is NULL, or <CB variable>.CATEGORY is INPROGRESS (INVALID OPERAND error).
- <CB variable>.TEXT is NULL (EMPTY SOURCE error).
- STATION.RESULT.TEXT is not NULL (OCCUPIED DESTINATION error).

The <CB-to-CB> form of the MOVETEXT statement takes the buffer associated with <source CB variable>.TEXT, associates this buffer with <destination CB variable>.TEXT, and then assigns NULL to <source CB variable>.TEXT. The following error conditions result in fault-termination of the process:

- Either <source CB variable>.CATEGORY or <destination CB variable>.CATEGORY is INPROGRESS (INVALID OPERAND error).
- <source CB variable>.TEXT is NULL (EMPTY SOURCE error).
- <destination CB variable>.TEXT is not NULL (OCCUPIED DESTINATION error).

Syntax

<movetext statement>

— MOVETEXT — (

<request-to-CB>
<CB-to-request>
<CB-to-result>
<CB-to-CB>

) —————|

<request-to-CB>

— REQUEST — , —<CB variable>—————|

<CB-to-request>

—<CB variable>— , — REQUEST —————|

<CB-to-result>

—<CB variable>— , — RESULT —————|

<CB-to-CB>

—<source CB variable>— , —<destination CB variable>—————|

<source CB variable>

—<CB variable>—————|

<destination CB variable>

—<CB variable>—————|

Examples

MOVETEXT (REQUEST,OUTPUTCB)

MOVETEXT (OUTPUTCB,REQUEST)

MOVETEXT (INPUTCB,RESULT)

MOVETEXT (INPUTCB,OUTPUTCB)

NEWRESULT Statement

The NEWRESULT statement obtains a buffer for the nontext portion of an input result, associates this buffer with STATION.RESULT, and assigns NULL to *STATION.RESULT.TEXT*. If the station reference variable is NULL, a NULL STATION error occurs; if STATION.RESULT is not NULL, an OCCUPIED DESTINATION error occurs. In either case, the process is fault-terminated.

Syntax

<newresult statement>

— NEWRESULT —————|

NEXTREQUEST Statement

The NEXTREQUEST statement removes from the queue the head output request from the host system for the station indicated by the station reference variable and associates this request with *STATION.REQUEST*. Then, if an editor is specified for the station, the executive initiates an activation of the editor output process and gives the process access to the output message, STATION structure, and editor global data. The line control process is made to wait for the editor output process to complete, at which time *STATION.REQUEST.TEXT* contains the edited output text. If the editor output process fails, the line control process is fault-terminated. In addition, the following error conditions result in fault-termination of the line control process:

- The station reference variable is NULL (NULL STATION error).
- *STATION.REQUEST* is not NULL (OCCUPIED DESTINATION error).
- *STATION.QUEUED* is FALSE (EMPTY SOURCE error).

Syntax

<nextrequest statement>

— NEXTREQUEST —————|

NEXTSTATION Statement

The NEXTSTATION statement changes the station reference variable. Any eligible station in the line station list specified for the line can be referenced by the station reference variable. A station is eligible if STATION.READY is TRUE and if the station belongs to a stationset, if both the line and the stationset are unassociated or if the line and stationset are associated with each other. Both forms of the NEXTSTATION statement are restricted to considering only those stations that meet these criteria.

Syntax

<nextstation statement>

— NEXTSTATION — <select part> —

<select part>

— <Boolean expression> —

Explanation

If the select part is not specified, the NEXTSTATION statement assigns to the station reference variable the next eligible STATION structure for the line. If the station reference variable is currently pointing to the last station on the line, NEXTSTATION assigns NULL to the station reference variable. If the station reference variable was NULL, NEXTSTATION assigns to it the first eligible STATION structure on the line.

If the select part is specified, NEXTSTATION assigns to the station reference variable the next eligible STATION structure for which the select part has the value TRUE, starting with the STATION structure following the currently referenced one (or, if the station reference variable is NULL, starting with the first station on the line). If the station belongs to a stationset and the line was unassociated, the line and stationset are associated when a match is found. If a match is not found after the last STATION structure has been tested, the station reference variable is assigned the value NULL. The order in which the NEXTSTATION statement returns a reference to a station can change if stations are deleted from and then added to a line.

Example

```
NEXTSTATION STATION.USEFORMS AND NOT CHANGEPENDING
```

NULLSTATION Statement

The NULLSTATION statement assigns the value NULL to the station reference variable.

The state of a station cannot be changed if any references to the station exist. In a multidrop line, use of the NEXTSTATION statement releases the reference to the current station and advances the reference to the next station. In the case where only one station is on a line, the NULLSTATION statement should be used periodically; otherwise, the station cannot be made not ready.

Syntax

<nullstation statement>

— NULLSTATION —————|

PROCESS Statement

The PROCESS statement changes the state of the specified event variable to NOT HAPPENED, places the specified process in execution, associates the event variable with the process invocation, and then returns to the next statement without waiting for the process to complete. (The WAIT statement or WAIT function can be used to wait for process completion.) When the process terminates, the executive causes the event variable associated with the process invocation. PROCESS statements can appear only within the LINE process of line control.

Syntax

<process statement>

— PROCESS — (—<process ID>— , —<event variable>—) —————|

Example

PROCESS(DIALIT,MYEVENT)

PURGE Statement

If REQUESTLIST is specified, the PURGE statement discards the requests specified by the purge selector in STATION.REQUESTLIST. If an asterisk (*) is specified or if the value of the key is greater than 255, all requests are discarded. If the station reference variable is NULL, a NULL STATION error occurs; if a key is specified and no request in STATION.REQUESTLIST is associated with the key, an EMPTY SOURCE error occurs. In either case, the process is fault-terminated.

If RESULTLIST is specified, the PURGE statement discards the results specified by the purge selector in STATION.RESULTLIST. If an asterisk (*) is specified, all results are discarded. If the station reference variable is NULL, a NULL STATION error occurs; if a key is specified and no result in STATION.RESULTLIST is associated with the key, an EMPTY SOURCE error occurs. In either case, the process is fault-terminated.

Syntax

<purge statement>

— PURGE — (

REQUESTLIST
RESULTLIST

 , —<purge selector>—) —————|

<purge selector>

<key>
*

 —————|

<key>

—<integer expression>—————|

REBLOCK Statement

The REBLOCK statement reblocks characters in the buffer that is associated with <CB variable>.TEXT. These characters are assumed to have been blocked based on the character size of the translate table specified by the line class code attribute and are reblocked 8 bits per character. The <residue in> variable specifies the number of residue bits in the last text character. The variable indicated by <residue out> is assigned the value of the residue after reblocking. Then, <CB variable>.TEXTSIZE is updated to reflect the actual number of text bytes after reblocking.

If <CB variable>.CATEGORY is INPROGRESS, or if <residue in> is greater than or equal to the original blocking factor, an INVALID OPERAND error occurs. If <CB variable>.TEXT is NULL, an EMPTY SOURCE error occurs. In any of these cases, the process is fault-terminated.

Syntax

```
<reblock statement>
— REBLOCK — ( —<CB variable>— , —<residue in>— , —<residue out>—→
→ ) —————|
<residue in>
—<integer expression>—————|
<residue out>
—<byte variable>—————|
  └<integer variable>┘
```

SENDBHOST Statement

The SENDHOST statement does the following:

1. Takes the result associated with STATION.RESULT
2. Assigns the type of the result either INPUT or ERROR as specified by the result type
3. Places the result in a queue for the executive
4. Assigns the value NULL to STATION.RESULT

If the station reference variable is NULL, a NULL STATION error occurs; if STATION.RESULT is NULL, an EMPTY SOURCE error occurs. In either case, the process is fault-terminated.

If INPUT is specified, executive processing of the result is controlled by STATION.INPUTACTION. If STATION.INPUTACTION is SENDANDWAIT, the SENDHOST statement suspends the process.

If ERROR is specified, the executive unconditionally sends the result to the host, assigns FALSE to STATION.READY, and assigns TRUE to LINE.STATUSCHANGED for all lines that have visibility of the station.

Syntax

<sendhost statement>

— SENDHOST —<result type>—————|

<result type>

— [ERROR
INPUT] —————|

TRACE Statement

The TRACE statement copies the value of the string expression into the result message, changes the type of the result to TRACEINPUT, and enqueues the result for the executive. No station or line variables are altered by this statement.

Syntax

<trace statement>

— TRACE —<string expression>—————|

UNLOCK Statement

The UNLOCK statement releases (unlocks) the specified lock. If the specified lock either is not locked or was locked by another process, an INVALID KERNEL CALL error occurs, and the process is fault-terminated. If any processes are suspended waiting for the lock, the first process that attempted to obtain (lock) the lock locks it, and any other suspended processes remain suspended.

Syntax

```
<unlock statement>
— UNLOCK —<lock variable>—————|
```

Example

```
UNLOCK LINE.TABLELOCK
```

WAIT Statement

If the WAIT statement specifies an integer expression, the process is suspended for at least the number of milliseconds specified by the value of the integer expression (modulo 32768). If the WAIT statement specifies a wait event, the state of the wait event is interrogated. If the state is HAPPENED, the state is changed to NOT HAPPENED; otherwise, the process is suspended until the wait event is caused.

Syntax

```
<wait statement>
— WAIT —<integer expression>—————|
      |<wait event>—————|
<wait event>
—<CB variable>—————|
  |<event variable>—————|
```

Explanation

The following information describes the wait events:

Wait Event	Cause
<CB variable>	This implicit event is caused by the executive when the operation that was initiated with the specified control block terminates.
<event variable>	This event is caused when another process executes a CAUSE statement for the specified event or, if the event was associated with a process invocation, when that process terminates.

Example

```
WAIT SYNCEVENT
```

Special Line Control Functions

The HAPPENED function, NULLREFERENCE function, TIMEINTERVAL function, and WAIT function, which follow, are valid only in line control. In addition, all special editor functions are allowed in line control.

HAPPENED Function

The HAPPENED function is a Boolean function that is equal to TRUE if the state of the wait event is HAPPENED and FALSE if the state of the wait event is NOT HAPPENED. Refer to “WAIT Statement” in this section for a description of the wait event.

Syntax

```
<happened function>  
— HAPPENED — ( —<wait event>— ) —————|
```

Example

```
DONE := HAPPENED(FINISHED)
```

NULLREFERENCE Function

The NULLREFERENCE function is a Boolean function that is equal to TRUE if the reference variable is NULL and FALSE if the reference variable is not NULL. If STATIONREFERENCE is specified, the station reference variable is tested.

Syntax

```
<nullreference function>  
— NULLREFERENCE — ( —<reference variable>— ) —————|  
<reference variable>  
— <CB variable>.TEXT —————|  
— STATIONREFERENCE —————|  
— STATION.REQUEST —————|  
— STATION.REQUEST.TEXT —————|  
— STATION.RESULT —————|  
— STATION.RESULT.TEXT —————|
```

TIMEINTERVAL Function

The TIMEINTERVAL function, given a valid string expression as defined below, returns an integer value that is the interval of time in seconds, rounded to the nearest second, between the specified timestamp string and the current value of the predeclared line control variable TIMESTAMP.

Syntax

```
<timeinterval function>  
— TIMEINTERVAL — ( —<timestamp string>— ) —————|  
<timestamp string>  
—<string expression>—————|
```

Explanation

The timestamp string is assumed to conform to TIMESTAMP in structure and content. (Refer to “Predeclared LINE Structure Variables” in this section.) If the length of the timestamp string is not equal to 8 when the function is evaluated, or if any of the eight integers that compose the timestamp string are not in the range of values specified for TIMESTAMP, an INVALID OPERAND error occurs, and the process is fault-terminated.

The TIMEINTERVAL function correctly handles time intervals that span different dates. However, if the time interval to be returned is larger than 65535 seconds, the value returned is exactly 65535.

Example

```
DELTA := TIMEINTERVAL(OLDTIME)
```

WAIT Function

The WAIT function interrogates the state of successive wait events, starting with the leftmost wait event. Refer to "WAIT Statement" in this section for a description of wait events.

Syntax

```

<wait function>
— WAIT — ( —<wait parameter list>— ) —————|
<wait parameter list>
—<wait timeout>— , ————|—————|
      |—————|/254\|—————|<wait event>|
<wait timeout>
|—————|<integer expression>|—————|
|—————|*|—————|

```

Explanation

If a wait event with a state of HAPPENED is encountered, the state of that wait event is changed to NOT HAPPENED, and the WAIT function returns the position of that wait event, starting with 1, within the wait parameter list.

If no wait event with a state of HAPPENED is encountered, the process is suspended until either one of the wait events is caused or the wait timeout elapses. The wait timeout parameter specifies the maximum length of time that the process is to be suspended while waiting for one of the wait events to be caused. If an integer expression is specified, the WAIT function returns the value 0 if none of the wait events has been caused within <integer expression> (modulo 32768) milliseconds, and the process resumes execution. If an asterisk (*) is specified, the process remains suspended indefinitely, and the function never returns the value 0.

Examples

```
EVENTINDEX := WAIT(1000,CHANGEVENT,SYNCEVENT)
```

```
T1 := WAIT(*,XCB,LINE.SYSTEM)
```

In the second example above, if XCB is caused, then T1 is assigned the value 1. If LINE.SYSTEM is caused, then T1 is assigned the value 2.

Line Control Fault Termination

When a line control process, or an editor process invoked by that line control process, commits an irrecoverable error or executes an ABORT statement, all processes in that line control are terminated, and all locks that were locked by any of those processes are unlocked. In addition, a result is returned to the host system specifying the line and station (if applicable) on which the error occurred and the nature of the fault. The host system must reinitialize the line for line activity to resume.

Adaptor Control

An adaptor control defines an INPUT and an OUTPUT process that implements the real-time, low-level aspects of a line-control protocol by suitable control of a line adaptor.

When a line is added to the data comm subsystem as the result of a host request, the executive does the following:

- Clears and then initializes the hardware of the assigned line adaptor
- Initializes the RECEIVER and TRANSMITTER structures associated with the line adaptor
- Allocates and initializes the adaptor control global data for the algorithm associated with the line (including the adaptor control copy of all duplicate structures).

The adaptor control processes do not execute continuously but are activated under the control of line control processes. When a line control process executes an INITIATE statement for an input control block, the executive activates the adaptor control INPUT process and gives this process access to the appropriate RECEIVER structure, TRANSMITTER structure, control block data, and adaptor control global data. When a line control process executes an INITIATE statement for an output control block, the executive activates the adaptor control OUTPUT process and gives this process access to the appropriate RECEIVER structure, TRANSMITTER structure, control block data, and adaptor control global data.

ADAPTOR CONTROL Definition

An ADAPTOR CONTROL section must include an INPUT process and an OUTPUT process. If an adaptor body is declared, these processes (and, optionally, additional declarations) appear in the NDLII program itself. If EXTERNAL is specified in place of an adaptor body, ADAPTOR CONTROL functions (including the INPUT and OUTPUT processes) are provided externally.

Syntax

```

<adaptor control>
┌── BIT ───┐ ADAPTOR CONTROL ─<adaptor control ID>──┐
└── CHARACTER ┘

→ ┌──<adaptor body>──┐
  └── EXTERNAL ───┘

<adaptor control ID>
──<identifier>──┐

<adaptor body>
── BEGIN ┌──<adaptor declaration list>──┐
          └── ; ───────────────────┘

→ ┌──┐ ┌──<input process declaration>──┐ ┌──┐ ┌── END ───┐
  └──┘ └──<output process declaration>──┘ └──┘ └── ; ───┘

<adaptor declaration list>
┌──┐ ┌── ; ───────────────────┐
└──┘ ┌──<define declaration>──┐
    └──<enumeration declaration>──┐
    └──<structure declaration>──┐
    └──<variable declaration list>──┘

```

Explanation

The DEFINE declaration, ENUMERATION declaration, STRUCTURE declaration, and variable declaration list are described in Section 2.

The INPUT process declaration is a PROCESS declaration for which the process ID is INPUT.

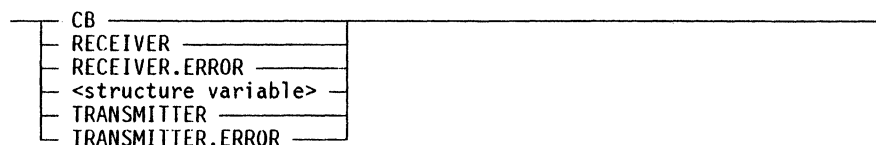
The OUTPUT process declaration is a PROCESS declaration for which the process ID is OUTPUT.

Adaptor Control Declarations

Adaptor control has access to variables in structure variables declared locally and in the adaptor control copy of duplicate structures declared in the algorithm in which the adaptor control appears. In addition, the structures CB, RECEIVER, and TRANSMITTER are predeclared in adaptor control. Variables in these structures are accessed by qualifying the variable name with the name of the structure.

Syntax

<adaptor control structure name>



Explanation

The CB structure is local to each process and corresponds to the control block that caused the adaptor control process to be initiated. The signal information in the control block is assigned by the line control process; the reply information can be assigned by the adaptor control process and returned to the line control process. The RECEIVER and TRANSMITTER structures are shared by both adaptor control processes. Care should be taken with shared variables, because no interlocking mechanism is provided.

The predeclared variables in the CB, RECEIVER, and TRANSMITTER structures are described in the information that follows.

Predeclared CB Structure Variable

The TEXT reference variable is a CB structure variable that is predeclared in adaptor control. TEXT is a pointer that references the text buffer area associated with the control block.

Predeclared RECEIVER Structure Variables

The RECEIVER structure is used to control and communicate with the receiver portion of the transceiver. The following variables are predeclared in the RECEIVER structure.

BCS (Integer, Read/Write)

BCS contains the receive BCS that is accumulated by the firmware. BCS does not contain the frame check sequence (FCS), which is directly accumulated by the receiver in bit mode.

DLEDETECT (Boolean, Read-Only)

If RECEIVER.TRANSPARENT is TRUE in synchronous mode, a TRUE value for DLEDETECT indicates the following of the character in RECEIVER.LINECHAR:

- It was not one of the BCS characters.
- It was immediately preceded by a character that matched the DLE character specified by the line class SYNCCONTROL attribute.
- It was not included in the receive BCS.
- It was discarded.

DLEDETECT is initialized to FALSE by the ENABLE RECEIVER statement and is recomputed for each received character.

ECHOPLEX (Boolean, Read/Write)

In asynchronous mode, if ECHOPLEX is TRUE, the transceiver automatically echoes received characters. ECHOPLEX is initialized to FALSE. If asynchronous mode is not specified in the line class of the line, any attempt to modify ECHOPLEX acts as a SKIP statement.

ENABLED (Boolean, Read-Only)

If ENABLED is TRUE, the receiver is enabled. ENABLED is initialized to FALSE and is affected by the ENABLE statement and DISABLE RECEIVER statement.

ERROR (Structure, Read-Only)

ERROR indicates receive errors. The individual RECEIVER.ERROR variables are defined as follows:

Variable	Description
ADDRESSERROR (Boolean)	In bit mode, ADDRESSERROR is assigned the value TRUE if the maximum length of a string variable that is the destination of the address field of a frame is not large enough to hold all of the address field. ADDRESSERROR is initialized to FALSE by ENABLE RECEIVER and INITIALIZE RECEIVER.ERROR statements.
BCSERROR (Boolean)	BCSERROR is assigned the value TRUE to indicate that a horizontal parity error was detected. BCSERROR is initialized to FALSE by the ENABLE RECEIVER and INITIALIZE RECEIVER.ERROR statements.
DATASETNOTREADY (Boolean)	DATASETNOTREADY is assigned the value TRUE if the DATA SET READY signal was initially FALSE or became FALSE during reception. DATASETNOTREADY is initialized to the opposite of the DATA SET READY signal state of the line by both the ENABLE RECEIVER and INITIALIZE RECEIVER.ERROR statements.
DISCONNECT (Boolean)	DISCONNECT is assigned the value TRUE if a switched line was initially disconnected or became disconnected during reception. DISCONNECT is initialized to the opposite of the connection state of the line by the ENABLE RECEIVER and INITIALIZE RECEIVER.ERROR statements.
FORMATERROR (Boolean)	FORMATERROR is assigned the value TRUE to indicate that a received character did not match the specified expected character. FORMATERROR is initialized to FALSE by the ENABLE RECEIVER and INITIALIZE RECEIVER.ERROR statements.
FRAMEABORT (Boolean)	In bit mode, FRAMEABORT is assigned the value TRUE to indicate that a forward frame abort was detected. In asynchronous mode, FRAMEABORT is assigned the value TRUE to indicate that the receiver detected a line break. FRAMEABORT is initialized to FALSE by the ENABLE RECEIVER and INITIALIZE RECEIVER.ERROR statements.
LOSSOFCARRIER (Boolean)	LOSSOFCARRIER is assigned the value TRUE if the carrier signal became FALSE during reception. LOSSOFCARRIER is initialized to FALSE by the ENABLE RECEIVER and INITIALIZE RECEIVER.ERROR statements.
OVERRUN (Boolean)	OVERRUN is assigned the value TRUE to indicate that at least one received character was lost because the receiver holding register was not unloaded in time. OVERRUN is initialized to FALSE by the ENABLE RECEIVER and INITIALIZE RECEIVER.ERROR statements.
PARITY (Boolean)	PARITY is assigned the value TRUE to indicate that a vertical parity error was detected. PARITY is initialized to FALSE by the ENABLE RECEIVER and INITIALIZE RECEIVER.ERROR statements.
STOPBIT (Boolean)	In asynchronous mode, STOPBIT is assigned the value TRUE to indicate that a character-framing error was detected. STOPBIT is initialized to FALSE by the ENABLE RECEIVER and INITIALIZE RECEIVER.ERROR statements.
TIMEOUT (Boolean)	TIMEOUT is assigned the value TRUE to indicate that a receive timeout was detected. TIMEOUT is initialized to FALSE by both the ENABLE RECEIVER and INITIALIZE RECEIVER.ERROR statements.

EXTEND (CODE_EXT_TYPE, Read/Write)

The EXTEND variable is used by the firmware to decide what mode of code extension is currently in effect. The possible options are EXT_NONE, EXT_SO, and EXT_ESC_SO. EXT_NONE indicates that there is no code extension in use. EXT_SO specifies that extended characters are bracketed by the control characters SO (shift out) and SI (shift in). EXT_ESC_SO indicates that extended characters are bracketed by the control characters ESC SO and ESC SI.

Because code extension cannot be used in conjunction with transparency, do not assign EXTEND any value except EXT_NONE while TRANSPARENT is true; any other value will result in an INVALID OPERAND error during execution.

Do not interrogate or manipulate the EXTEND variable on a line that runs on an NSP/LSP because EXTEND has not been implemented with NSP/LSP. Doing so results in an INVALID OPERAND error during execution of the program.

Note: When the EXTEND variable is assigned the value EXT_SO, the sole function of the characters SO and SI is that of control characters that indicate the presence of extended characters. Under these circumstances, SO and SI cannot be used to indicate reverse video or the underlining of characters.

FIRSTERROR (RTERROR, Read-Only)

The value of FIRSTERROR indicates which RECEIVER.ERROR, if any, was detected first. If more than one error is detected at the same time, the error whose enumerated constant has the greatest map value is indicated. FIRSTERROR is initialized to NONE by the ENABLE RECEIVER and INITIALIZE RECEIVER.ERROR statements.

FIRSTERROR can be assigned only during the execution of a RECEIVE statement. If a spontaneous receive error (for example, a loss of carrier) is detected outside the execution of a RECEIVE statement, the appropriate RECEIVER.ERROR Boolean variable is assigned; if FIRSTERROR is to be assigned, it is not assigned until the next RECEIVE statement is executed. Thus, spontaneous RECEIVE errors that are detected between the final RECEIVE statement and a DISABLE RECEIVER statement cannot cause FIRSTERROR to be assigned.

IDLEDETECT (Boolean, Read-Only)

In bit mode, a TRUE value for IDLEDETECT indicates that the receiver has received at least 15 consecutive one-bits. If IDLEDETECT is TRUE and the receiver detects a zero-bit, IDLEDETECT is assigned the value FALSE.

INHIBIT (Boolean, Read-Only)

A TRUE value for INHIBIT indicates that at least one of the following RECEIVER.ERROR Boolean variables is TRUE:

- DATASETNOTREADY
- DISCONNECT
- FRAMEABORT
- LOSSOFCARRIER
- TIMEOUT

INHIBIT is initialized to the appropriate value by ENABLE RECEIVER and INITIALIZE RECEIVER.ERROR statements.

LINECHAR (Byte, Read-Only)

LINECHAR contains the last character unloaded from the receiver holding register (before translation).

RESIDUE (Integer, Read/Write)

In bit mode, if the count of the number of I-field bits in the last frame received is given by N and the character size of the translate table specified by the line class code attribute is given by M, then RESIDUE is assigned the value $(N \text{ MOD } M)$. If RESIDUE is nonzero, the last I-field character assembled by the receiver contains the residue bits right-justified with zero fill.

SPECIAL (Boolean, Read-Only)

The value of SPECIAL indicates the current value of the special input signal on the electrical interface for the line. The field engineer is responsible for strapping this signal to the desired input signal on the line adaptor. This feature is available only on NSPs/LSPs.

TRANSLATE (Boolean, Read/Write)

If TRANSLATE is TRUE, translation of received characters is enabled. TRANSLATE is initialized to TRUE.

TRANSPARENT (Boolean, Read/Write)

In synchronous mode, if TRANSPARENT is TRUE, the receiver is enabled for transparent operation. TRANSPARENT is initialized to FALSE. If synchronous mode is not specified in the class of the line, any attempt to modify TRANSPARENT acts as a SKIP statement.

Predeclared TRANSMITTER Structure Variables

The TRANSMITTER structure is used to control and communicate with the transmitter portion of the transceiver. The following variables are predeclared in the TRANSMITTER structure.

BCS (Integer, Read/Write)

BCS contains the transmit BCS that is accumulated by the firmware. BCS does not contain the frame check sequence (FCS), which is directly accumulated by the transmitter in bit mode.

DIAGNOSE (Boolean, Read/Write)

When DIAGNOSE is assigned the value TRUE, the receiver and transmitter are internally tied in a self-test mode and are disconnected from the data comm line. The DATA SET READY signal is tied to the DATA TERMINAL READY signal, which is internally held TRUE. The CARRIER and CLEAR TO SEND signals are tied to the REQUEST TO SEND signal. Entry into diagnostic mode results in the loss of a switched connection; the disconnection is not noticed until diagnostic mode is disabled.

ENABLED (Boolean, Read-Only)

If ENABLED is TRUE, the transmitter is enabled. ENABLED is initialized to FALSE and is changed by the ENABLE TRANSMITTER and DISABLE TRANSMITTER statements.

ERROR (Structure, Read-Only)

ERROR indicates transmit errors. The individual TRANSMITTER.ERROR variables are defined as follows:

Variable	Description
ADDRESSERROR (Boolean)	In bit mode, ADDRESSERROR is assigned the value TRUE if a string expression that is the source of the address field of a frame did not contain a well-formed address field. ADDRESSERROR is initialized to FALSE by the ENABLE TRANSMITTER and INITIALIZE TRANSMITTER.ERROR statements.
DATASETNOTREADY (Boolean)	DATASETNOTREADY is assigned the value TRUE to indicate that the DATA SET READY signal either was initially FALSE or became FALSE during transmission. DATASETNOTREADY is initialized to the opposite of the DATA SET READY signal state of the line by the ENABLE TRANSMITTER and INITIALIZE TRANSMITTER.ERROR statements.
DISCONNECT (Boolean)	DISCONNECT is assigned the value TRUE to indicate that a switched line either was initially disconnected or became disconnected during transmission. DISCONNECT is initialized to the opposite of the connection state of the line by both the ENABLE TRANSMITTER and INITIALIZE TRANSMITTER.ERROR statements.

continued

continued

Variable	Description
RESIDUEERROR (Boolean)	In bit mode, RESIDUEERROR is assigned the value TRUE if the number of residue bits specified by TRANSMITTER.RESIDUE for a frame is greater than or equal to the number of bits per I-field character indicated by the TRANSMIT FRAME statement. RESIDUEERROR is initialized to the value FALSE by the ENABLE TRANSMITTER and INITIALIZE TRANSMITTER.ERROR statements.
TIMEOUT (Boolean)	TIMEOUT is assigned the value TRUE to indicate that a transmit timeout was detected. TIMEOUT is initialized to FALSE by the ENABLE TRANSMITTER and INITIALIZE TRANSMITTER.ERROR statements.
UNDERRUN (Boolean)	In bit mode, UNDERRUN is assigned the value TRUE to indicate that a forward frame abort was automatically performed because the transmitter holding register was not unloaded in time. UNDERRUN is initialized to FALSE by the ENABLE TRANSMITTER and INITIALIZE TRANSMITTER.ERROR statements.

EXTEND (CODE_EXT_TYPE, Read/Write)

The EXTEND variable is used by the firmware to decide what mode of code extension is currently in effect. The possible options are EXT_NONE, EXT_SO, and EXT_ESC_SO. EXT_NONE indicates that there is no code extension in use. EXT_SO specifies that extended characters are bracketed by the control characters SO and SI. EXT_ESC_SO indicates that extended characters are bracketed by the control characters ESC SO and ESC SI.

Because code extension cannot be used in conjunction with transparency, do not assign EXTEND any value except EXT_NONE while TRANSPARENT is true; any other value will result in an INVALID OPERAND error during execution.

Do not interrogate or manipulate the EXTEND variable on a line that runs on an NSP/LSP because EXTEND has not been implemented with NSP/LSP. Doing so results in an INVALID OPERAND error during execution of the program.

Note: When the EXTEND variable is assigned the value EXT_SO, the sole function of the characters SO and SI is that of control characters that indicate the presence of extended characters. Under these circumstances, SO and SI cannot be used to indicate reverse video or the underlining of characters.

FIRSTERROR (RTERROR, Read-Only)

The value of FIRSTERROR indicates which TRANSMITTER.ERROR, if any, is detected first. If more than one error is detected at the same time, the error whose enumerated constant has the greatest map value is indicated. FIRSTERROR is initialized to NONE by the ENABLE TRANSMITTER and INITIALIZE TRANSMITTER.ERROR statements.

FIRSTERROR can be assigned only during the execution of a **TRANSMIT** statement, **ENABLE TRANSMITTER** statement, or **DISABLE TRANSMITTER** statement. If a spontaneous transmit error such as **DATA SET NOT READY** is detected outside the execution of one of these statements, the appropriate **TRANSMITTER.ERROR** Boolean variable is assigned the value **TRUE**; if **FIRSTERROR** is to be assigned, it is not assigned until the next such statement is executed.

INHIBIT (Boolean, Read-Only)

A **TRUE** value for **INHIBIT** indicates that either **FETCHPARITY** or one of the **TRANSMITTER.ERROR** Boolean variables is **TRUE**. **INHIBIT** is initialized to the appropriate value by the **ENABLE TRANSMITTER** and **INITIALIZE TRANSMITTER.ERROR** statements.

LINECHAR (Byte, Read-Only)

LINECHAR contains the last character loaded into the transmitter-holding register (after translation).

RESIDUE (Integer, Read/Write)

In bit mode, if the count of the number of I-field bits in the next frame to be transmitted is given by N and the character size of the translate table specified by the line class **CODE** attribute is given by M , then **RESIDUE** must be assigned the value $(N \text{ MOD } M)$. If **RESIDUE** is not zero, the last I-field character must contain the residue bits right-justified with zero fill.

SPECIAL (Boolean, Read/Write)

SPECIAL is used to control the special output signal on the electrical interface for the line. **SPECIAL** is initialized to **FALSE**. The field engineer is responsible for strapping this signal to the desired output signal on the line adaptor. This feature is available only on NSPs/LSPs.

TRANSLATE (Boolean, Read/Write)

If **TRANSLATE** is **TRUE**, translation of transmitted characters is enabled. **TRANSLATE** is initialized to **TRUE**.

TRANSPARENT (Boolean, Read/Write)

In synchronous mode, if **TRANSPARENT** is **TRUE**, the transmitter is enabled to enter transparent mode after execution of a **TRANSMIT** statement with the **FORCE DLE** option. **TRANSPARENT** is initialized to **FALSE**. If synchronous mode is not specified in the class of the line, any attempt to modify **TRANSPARENT** acts as a **SKIP** statement.

Adaptor Control Predeclared Variables

The following variables are predeclared in, and local to, each adaptor control process.

BUFFEROVERFLOW (INPUT Process Only; Boolean, Read-Only)

If **BUFFEROVERFLOW** is **TRUE**, an attempt was made to store past the end of the buffer. Each time the process is initiated, **BUFFEROVERFLOW** is initialized to the appropriate value.

ENDTEXT (OUTPUT Process Only; Boolean, Read-Only)

If **ENDTEXT** is **TRUE**, either all the characters have been fetched from the text buffer associated with the control block, or no text buffer is associated with the control block. Each time the process is initiated, **ENDTEXT** is initialized to the appropriate value.

FETCHPARITY (Boolean, Read-Only)

FETCHPARITY is assigned the value **TRUE** if a parity error was detected during the transfer of text characters or of a duplicate structure value from line control to the process. Each time the process is initiated, **FETCHPARITY** is initialized to **FALSE**.

NOCANCEL (Boolean, Read/Write)

The value of **NOCANCEL** controls the action of a **CANCEL** statement executed by the other adaptor control process and the action of a line control **INITIATE CANCEL** statement. Each time the process is initiated, **NOCANCEL** is initialized to **FALSE**.

Adaptor Control Process Declarations

Two processes must be declared in adaptor control: one with a process ID of **OUTPUT** and one with a process ID of **INPUT**.

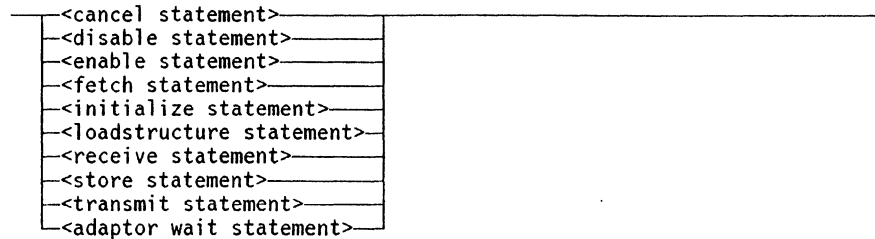
The **OUTPUT** process is activated when a process in line control initiates an output control block by executing an **INITIATE** statement. Similarly, the **INPUT** process is activated when a process in line control initiates an input control block by executing an **INITIATE** statement.

Special Adaptor Control Statements

In addition to the general statements, the statements that follow are valid in adaptor control. The special adaptor statements are described in the information that follows the general syntax diagram.

Syntax

<special adaptor statement>



CANCEL Statement

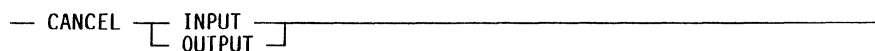
The CANCEL statement is used both to wait for and to conditionally force termination of all line-control-initiated operations involving the other adaptor control process. The specified process name must be that of the other adaptor control process (for example, a CANCEL statement in the OUTPUT process must specify INPUT).

If the other adaptor control process is inactive, the CANCEL statement acts as a SKIP statement.

If the other adaptor control process is active, the action of the CANCEL statement depends on the state of the NOCANCEL variable for the other adaptor control process. If NOCANCEL is FALSE, the other adaptor control process is immediately canceled (terminated with the CATEGORY variable of the control block assigned the value CANCELLED). If NOCANCEL is TRUE, the CANCEL statement waits until either NOCANCEL is assigned the value FALSE, at which time the other adaptor control process is canceled, or the other adaptor control process terminates for an unrelated reason. In any case, all line-control-initiated operations that are waiting until the previous invocation of the other adaptor control process terminates are canceled when the previous process terminates.

Syntax

<cancel statement>



DISABLE Statement

The **DISABLE RECEIVER** statement is used to disable the receiver. If **RECEIVER.ENABLED** is **FALSE**, a **RECEIVER NOT ENABLED** error occurs, and the process is fault-terminated. The disabling process includes assigning the value **FALSE** to **RECEIVER.ENABLED**.

The **DISABLE TRANSMITTER** statement is used to disable the transmitter. If **TRANSMITTER.ENABLED** is **FALSE**, a **TRANSMITTER NOT ENABLED** error occurs, and the process is fault-terminated. The disabling process includes assigning **FALSE** to **TRANSMITTER.ENABLED**. Due to internal buffering in the transmitter, the transmitter is not actually disabled until the last transmitted character has been placed on the line; transmission errors can occur during this time.

Syntax

<disable statement>

— **DISABLE** — **RECEIVER** —
 └ **TRANSMITTER** ┘

ENABLE Statement

The **ENABLE RECEIVER** statement is used to enable a receiver. If **RECEIVER.ENABLED** is **TRUE**, a **RECEIVER ALREADY ENABLED** error occurs, and the process is fault-terminated. The enabling process involves the following sequence:

1. **RECEIVER.DLEDETECT** is assigned the value **FALSE**.
2. **RECEIVER.RESIDUE** is assigned the value **0**.
3. **RECEIVER.ENABLED** is the value assigned **TRUE**.
4. The receiver is enabled.
5. A delay occurs for the number of milliseconds (modulo 32768) specified by the **RECEIVEDDELAY** of the line.
6. **RECEIVER.ERROR** is initialized.
7. **RECEIVER.FIRSTERROR** is initialized.
8. **RECEIVER.INHIBIT** is initialized.

The **ENABLE TRANSMITTER** statement is used to enable a transmitter. If **TRANSMITTER.ENABLED** is **TRUE**, a **TRANSMITTER ALREADY ENABLED** error occurs, and the process is fault-terminated. The enabling process involves the following sequence:

1. **TRANSMITTER.RESIDUE** is assigned the value **0**.
2. **TRANSMITTER.ENABLED** is assigned the value **TRUE**.
3. The transmitter is enabled.
4. A delay occurs for the number of milliseconds (modulo 32768) specified by the **TRANSMITDELAY** of the line.
5. **TRANSMITTER.ERROR** is initialized.
6. **TRANSMITTER.FIRSTERROR** is initialized.
7. **TRANSMITTER.INHIBIT** is initialized.

Syntax

<enable statement>

— **ENABLE** — RECEIVER
TRANSMITTER —

FETCH Statement

The **FETCH** statement obtains a character from the next position of the text buffer associated with the control block and stores the result in the specified byte variable. If **ENDTEXT** is **TRUE**, a **FETCH OVERFLOW** error occurs, and the process is fault-terminated. The **FETCH** statement is valid only in the **OUTPUT** process.

If **FETCHPARITY** is **TRUE** or if the last character in the text buffer is being fetched, **ENDTEXT** is assigned the value **TRUE**.

Syntax

<fetch statement>

— **FETCH** —<byte variable>—————|

Example

```
FETCH NEXTCHAR
```

INITIALIZE Statement

The **INITIALIZE RECEIVER.BCS** statement causes **RECEIVER.BCS** to be initialized to the initial value specified by the horizontal **PARITY** attribute in the line class of the line.

The **INITIALIZE RECEIVER.ERROR** statement causes the **RECEIVER.ERROR** structure, **RECEIVER.FIRSTERROR**, and **RECEIVER.INHIBIT** to be initialized.

The **INITIALIZE TRANSMITTER.BCS** statement causes **TRANSMITTER.BCS** to be initialized to the initial value specified by the horizontal **PARITY** attribute in the line class of the line.

The **INITIALIZE TRANSMITTER.ERROR** statement causes the **TRANSMITTER.ERROR** structure, **TRANSMITTER.FIRSTERROR**, and **TRANSMITTER.INHIBIT** to be initialized.

Syntax

<initialize statement>

— **INITIALIZE** —

—	RECEIVER.BCS	—————
—	RECEIVER.ERROR	—————
—	TRANSMITTER.BCS	—————
—	TRANSMITTER.ERROR	—————

LOADSTRUCTURE Statement

The LOADSTRUCTURE statement loads the value of the specified structure from the text buffer associated with the control block. The structure must have been declared in a DUPLICATE STRUCTURE declaration in the ALGORITHM declaration list for the algorithm in which this adaptor control section appears. If the structure is not the same structure that line control copied into the text buffer of the control block, the process is fault-terminated.

Syntax

<loadstructure statement>

— LOADSTRUCTURE —<structure>—

<structure>

—<structure ID>—
└─<structure array ID>—

Example

LOADSTRUCTURE STRUCTUREV

RECEIVE Statement

The RECEIVE statement is used to perform a sequence of receive actions. If RECEIVER.ENABLED is FALSE, a RECEIVER NOT ENABLED error occurs, and the process is fault-terminated.

Syntax

<receive statement>

```

— RECEIVE —<time part> —<receive item> —

```

<time part>

```

— ( — NOTIMEOUT — ) —
  | —<receive timeout> —
  | * —

```

<receive timeout>

```

—<integer expression> —

```

<receive item>

```

— BCS —
  | — FRAME — ( —<receive frame field part> — ) —
  | — TEXT —
  | —<string variable> — UNTIL —<receive terminate list> —
  | —<expected character> —
  | —<character> —

```

<receive frame field part>

```

—<receive address field> — , —<receive control field> — , —

```

```

→<receive I-field> —

```

<receive address field>

```

—<string variable> —

```

<receive control field>

```

—<integer variable> —

```

<receive I-field>

```

— TEXT —
  | —<string variable> —

```

<receive terminate list>

```

— OR —
  | — /1\—<character set> —
  | — /1\— DLEDETECT —
  | — /1\— BCSERROR —
  | — /1\— FORMATERROR —
  | — /1\— OVERRUN —
  | — /1\— PARITY —
  | — /1\— STOPBIT —

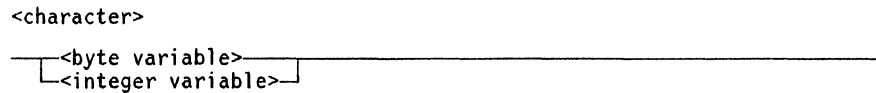
```

<expected character>

```

—<byte constant> —

```



Explanation

A time part variable specifies the maximum number of milliseconds (modulo 32768) that the statement waits for each subsequent received character. IF NOTIMEOUT is specified, a timeout value of 0 is used, implying an indefinite wait. If an asterisk (*) is specified, the line TIMEOUT attribute is used; if, in this case, the line TIMEOUT attribute has no value, the timeout value has a default value of 0, which, if used in the RECEIVE statement, is equivalent to specifying NOTIMEOUT.

Each receive item corresponds to a particular type of receive action. The receive items are described as follows:

Receive Item	Description
BCS	If horizontal parity is specified in the line class of the line and if RECEIVER.INHIBIT is FALSE, this receive item unloads the next characters from the receiver-holding register and compares them to the receive BCS. If the unloaded characters do not match the receive BCS, <i>RECEIVER.ERROR.BCSERROR</i> is assigned the value TRUE. This receive item is allowed only in character algorithms.
FRAME	If RECEIVER.INHIBIT is FALSE and neither end-of-frame nor frame abort is detected by the receiver, this receive item unloads successive characters from the receiver-holding register and either stores them in the specified field destinations (if RECEIVER.ERROR.OVERRUN is FALSE) or discards them. This receive item is allowed only in bit algorithms. Address field characters are either stored untranslating in the specified string starting with <string variable>[1] (if the maximum length of the string is not exceeded) or discarded (in which case, RECEIVER.ERROR.ADDRESSERROR is assigned TRUE). The length of the string is assigned a value that indicates the number of characters stored (which might be 0). If the line class ADDRESS MODE attribute of the line specifies BASIC, the address field consists of only 1 character; otherwise, the address field ends with the first address character containing a least significant bit of 1. Control field characters are stored untranslating in an integer. If the line class CONTROL MODE attribute of the line specifies BASIC, the control field consists of 1 character, which is stored right-justified with zero fill; otherwise, the control field consists of 2 characters, which are stored as follows: the first control character in the most significant byte and the second control character in the least significant byte.

Receive Item	Description
	Any I-field characters are stored either in a string or in the text buffer associated with the control block. If the destination is a string, characters are either stored in the specified string starting with <string variable>[1] (if the maximum length of the string is not exceeded) or discarded (in which case, RECEIVER.ERROR.FORMATERROR is assigned the value TRUE). The length of the string is assigned a value that indicates the number of characters stored (which might be 0). If the destination is TEXT, characters are either stored in the text buffer starting at the current buffer position (if the text buffer is not full) or discarded (in which case, the value TRUE is assigned to <i>RECEIVER.ERROR.FORMATERROR</i>). If RECEIVER.TRANSLATE is TRUE, each character is translated before being stored.
TEXT	If RECEIVER.INHIBIT is FALSE and if none of the specified RECEIVER.ERROR receive terminate list conditions, if any, are TRUE, this receive item unloads successive characters from the receiver-holding register and stores them, starting at the current buffer position, in the text buffer associated with the control block. (The receive terminate list is discussed later in this section.) If a receive terminate list is not specified, the operation terminates when the text buffer becomes full. If a receive terminate list without a character set is specified, the operation terminates when one of the specified conditions becomes TRUE (in which case, the last character received is not stored) or when the text buffer becomes full. If a receive terminate list with a character set is specified, the operation terminates as described previously, except when none of the specified conditions has been met and the text buffer becomes full. In this case, one final character is received and checked for membership in the character set. If this character is not a member, RECEIVER.ERROR.FORMATERROR is assigned the value TRUE; in any case, the character is not stored. If RECEIVER.TRANSLATE is TRUE, each character is translated before being stored and, if horizontal parity is specified in the line class of the line, each (untranslated) character is included in the receive BCS. If the line class of the line specifies synchronous mode, sync-fill characters are discarded and are not included in the receive BCS. Sync-fill characters are determined by the line class SYNCCONTROL attribute and the value of <i>RECEIVER.TRANSPARENT</i>. (If RECEIVER.TRANSPARENT is TRUE, the receiver recognizes DLE/SYN pairs as sync-fill.) This receive item is allowed only in character algorithms.

Receive Item	Description
<string variable>	If RECEIVER.INHIBIT is FALSE and if none of the specified RECEIVER.ERROR receive terminate list conditions are TRUE, this receive item unloads successive characters from the receiver-holding register and stores them in the specified string starting with <string variable>[1]. The length of the string is assigned a value that indicates the number of characters stored (which might be 0). (The receive terminate list is discussed later in this section.) If a receive terminate list is not specified, the operation terminates when the string becomes full. If a receive terminate list without a character set is specified, the operation terminates when one of the specified conditions becomes TRUE (in which case, the last character received is not stored) or when the string becomes full. If a receive terminate list with a character set is specified, the operation terminates as described previously, except when none of the specified conditions has been met and the string becomes full. In this case, one final character is received and checked for membership in the character set. If this character is not a member, RECEIVER.ERROR.FORMATERROR is assigned the value TRUE; in any case, the character is not stored. If RECEIVER.TRANSLATE is TRUE, each character is translated before being stored and, if horizontal parity is specified in the line class of the line, each (untranslated) character is included in the receive BCS. If the line class of the line specifies synchronous mode, sync-fill characters (as specified by the line class SYNCCONTROL attribute) are discarded and are not included in the receive BCS. This receive item is allowed only in character algorithms.
<expected character>	If RECEIVER.INHIBIT is FALSE, this receive item unloads the next character from the receiver-holding register and compares it to the specified byte constant; if they do not match, RECEIVER.ERROR.FORMATERROR is assigned the value TRUE. If RECEIVER.TRANSLATE is TRUE, the character is translated before the comparison and, if horizontal parity is specified in the class of the line, the (untranslated) character is included in the receive BCS. If the line class of the line specifies synchronous mode and the unloaded character is a sync-fill character (as specified by the line class SYNCCONTROL attribute), that character and all successive characters are discarded until a non-sync-fill character is unloaded. This receive item is allowed only in character algorithms.
<character>	This receive item either unloads the next character from the receiver-holding register and stores this character in the specified byte or integer variable (if RECEIVER.INHIBIT is FALSE), or stores 4"FFFF" in the specified byte or integer variable when timeout occurs. If RECEIVER.TRANSLATE is TRUE, the character is translated before being stored and, if horizontal parity is specified in the class of the line, the (untranslated) character is included in the receive BCS.

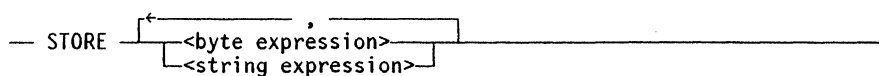
A receive terminate list specifies conditions that terminate a text or a string variable receive item. The DLEDETECT condition and the receive error Boolean conditions (BCSError, FORMATERror, OvErrun, PARITY, and STOPBIT) are satisfied by a TRUE value of the respective Boolean variable. DLEDETECT stands for RECEIVER.DLEDETECT, and each receive error Boolean variable stands for the corresponding RECEIVER.ERROR Boolean variable. A character set condition is satisfied when an unloaded (translated) character is a member of the character set. The character satisfying the character set condition is not stored, and the untranslated character is contained in RECEIVER.LINECHAR.

```
RECEIVE (NOTIMEOUT) TEMPSTRING UNTIL STX,  
(*) TEXT UNTIL ETX
```

The **STORE** statement places a sequence of characters in the next positions of the text buffer associated with a control block. This statement is valid only in the **INPUT** process.

No action is performed if BUFFER_OVERFLOW is TRUE. If an attempt is made to store past the end of the text buffer, BUFFER_OVERFLOW is assigned the value TRUE and the remaining characters in the sequence are skipped.

```
<store statement>
```



STORE LASTCHAR, CR & LF

TRANSMIT Statement

The TRANSMIT statement is used to perform a sequence of transmit actions. If TRANSMITTER.ENABLED is FALSE, a TRANSMITTER NOT ENABLED error occurs, and the process is fault-terminated.

Syntax

<transmit statement>

— TRANSMIT — [—<transmit item> —] —

<transmit item>

[—	<force DLE>	[—	BCS	—	]
			TEXT		
			UNTIL —<transmit termination list>		
			<byte expression>		
		<string expression>			
	BREAK				
			<break time>		
	FLAGS				
	FRAME		(—<transmit frame field part> —)		
	ONES				

<force DLE>

— [— DLE —] —

<transmit termination list>

[— OR —]
 [— /1\ —<character set> —]
 [— /1\ — BREAK —]

<break time>

—<integer expression>—

<transmit frame field part>

—<transmit address field>— , —<transmit control field>— , —

→<transmit I-field>— , —<I field blocking>—

<transmit address field>

—<string expression>—

<transmit control field>

—<integer expression>—

<transmit I-field>

— TEXT —
 [—<string expression>—]

<I-field blocking>

[8
 *]

Explanation

Each transmit item corresponds to a particular type of transmission action. If the transmitter fails to empty the transmitter-holding register within three seconds of an

attempt by a transmit item to load a new character into the transmitter-holding register, TRANSMITTER.ERROR.TIMEOUT is assigned the value TRUE.

The transmit items are described as follows:

Transmit Item	Description
<FORCE DLE>	As explained in the the discussion of the CLASS declaration, the FORCE DLE option is used in synchronous mode to control transparent operation. The first FORCE DLE option used after TRANSMITTER.TRANSPARENT is assigned the value TRUE, which causes the transmitter to enter transparent mode (DLE/SYN pairs are transmitted as sync-fill). Forced DLE characters are not included in the transmit BCS. The FORCE DLE option is ignored if TRANSMITTER.TRANSPARENT is FALSE or if synchronous mode is not specified in the line class of the line.
BCS	If horizontal parity is specified in the line class of the line and if TRANSMITTER.INHIBIT is FALSE, this transmit item loads the transmit BCS characters into the transmitter-holding register. This transmit item is allowed only in character algorithms.
TEXT	When TRANSMITTER.INHIBIT is FALSE, none of the transmit termination list conditions is met, and ENDTEXT is FALSE. Then this transmit item loads successive characters from the text buffer associated with the control block (starting at the current buffer position) into the transmitter-holding register. The transmit termination list is discussed later on in this section. If TRANSMITTER.TRANSLATE is TRUE, each character is translated before being loaded, and, if horizontal parity is specified in the line class of the line, each (translated) character is included in the transmit BCS. This transmit item is allowed only in character algorithms.
<byte expression>	If TRANSMITTER.INHIBIT is FALSE, this transmit item loads the specified character into the transmitter-holding register. If TRANSMITTER.TRANSLATE is TRUE, the character is translated before being loaded and, if horizontal parity is specified in the line class of the line, the (translated) character is included in the transmit BCS. This transmit item is allowed only in character algorithms.
<string expression>	If TRANSMITTER.INHIBIT is FALSE and the length of the specified string expression is not exceeded, this transmit item loads successive characters from the specified string expression (starting with <string expression>[1]) into the transmitter-holding register. If TRANSMITTER.TRANSLATE is TRUE, each character is translated before being loaded, and, if horizontal parity is specified in the line class of the line, each (translated) character is included in the transmit BCS. This transmit item is allowed only in character algorithms.

Transmit Item	Description
BREAK	If asynchronous mode is specified in the line class of the line and if TRANSMITTER.INHIBIT is FALSE, this transmit item places the line in a space condition. The line is held in a space condition until either another transmit item is performed or the transmitter is disabled; the space condition is held at least the number of milliseconds (modulo 32768) specified by a break time variable. If a break time is not specified, 2 character times is assumed. This transmit item is allowed only in character algorithms.
FLAGS	If TRANSMITTER.INHIBIT is FALSE, this transmit item initiates the continuous transmission of flags. Flags are transmitted until either another transmit item is performed or the transmitter is disabled; at least one flag is transmitted. This transmit item is allowed only in bit algorithms.
FRAME	While TRANSMITTER.INHIBIT is FALSE, this transmit item loads successive characters from the specified field sources into the transmitter-holding register. When the last character of the frame is loaded, an end-of-frame signal is given to the transmitter. This transmit item is allowed only in bit algorithms. While the length of the specified string expression is not exceeded, address field characters are loaded untranslated from the string expression (starting with <string expression>[1]). If the line class ADDRESS MODE attribute of the line specifies BASIC, the address field consists of just 1 character; otherwise, the address field ends with the first address character that has a least significant bit of 1. If the string expression does not contain a well-formed address field or if the length of the string specified as the address field is not exactly equal to the address length, TRANSMITTER.ERROR.ADDRESSERROR is assigned the value TRUE, and the frame is aborted. Control field characters are loaded untranslated from the specified INTEGER expression. If the line class CONTROL MODE attribute of the line specifies BASIC, the control field consists of 1 character, which is loaded from the least significant byte; otherwise, the control field consists of 2 characters, which are loaded as follows: the first control character from the most significant byte and the second control character from the least significant byte. I-field characters, if any, are loaded either from the specified string expression or from the text buffer associated with the control block. If the source is a string expression, consecutive characters are loaded starting with <string expression>[1] (while the length of the string expression is not exceeded). If the source is TEXT, consecutive characters are loaded starting at the current buffer position (if ENDTEXT is FALSE). If TRANSMITTER.TRANSLATE is TRUE, each character is translated before being loaded.

Transmit Item	Description
	If the I-field blocking parameter specifies an asterisk (*), I-field characters are transmitted based on the character size of the translate table specified by the line class CODE attribute. If "8" is specified, I-field characters are unconditionally transmitted 8 bits per character (which is useful for BDLC control frames with I-fields such as FRMR and XID). If the number of residue bits specified by TRANSMITTER.RESIDUE is greater than or equal to the number of bits per I-field character, the variable TRANSMITTER.ERROR.RESIDUEERROR is assigned the value TRUE, and the frame is aborted.
ONES	If TRANSMITTER.INHIBIT is FALSE, this transmit item initiates the continuous transmission of one-bits. One-bits are transmitted until either another transmit item is performed or the transmitter is disabled; at least 8 one-bits are transmitted. This transmit item is allowed only in bit algorithms.

A transmit termination list specifies conditions that terminate a TEXT transmit item. The BREAK condition is satisfied by a TRUE value of *RECEIVER.ERROR.FRAMEABORT*. A character set condition is satisfied when an (untranslated) character to be loaded is a member of the character set. The character satisfying the character set condition is not loaded, and the text buffer pointer is left pointing to this character. A FETCH statement can be used to retrieve the character at a later time.

Adaptor Control WAIT Statement

The adaptor WAIT statement delays process execution for at least the number of milliseconds (modulo 32768) specified by the value of the INTEGER expression.

Syntax

<adaptor wait statement>

— WAIT —<integer expression>—————|

Example

WAIT 200

Special Adaptor Control Functions

The BCSOF function, the adaptor TRANSLATE function, and the WAITFORIDLE function are valid only in adaptor control. They are described in the information that follows.

BCSOF Function

The BCSOF function allows the programmer to use the intrinsic BCS generator to calculate BCS polynomials independently of the calculations associated with the line by the CLASS declaration.

Syntax

```
<BCSOF function>
— BCSOF — ( — * ———— , —<integer expression>— , —————→
               |<polynomial>|
→<byte expression>— ) —————→
```

Explanation

This function returns the value generated by accumulating the character specified by the BYTE expression into the current value of the BCS specified by the INTEGER expression, using the given polynomial. If an asterisk (*) appears instead of a polynomial, the polynomial specified in the CLASS declaration is used. The syntax for the polynomial is described with the line class PARITY attribute under “Line Class Attributes” in this section.

The BCSOF function is computed by forming the exclusive OR of the least significant N bits (where N is the character size of the current translate table) of the new character with the most significant N bits of the current BCS, and then dividing the result (without carries) by the BCS polynomial. The remainder of this division is the new BCS.

Examples

```
BCSALL := BCSOF(*,BCSALL,DLE)
```

```
RECEIVER.BCS := BCSOF(X8+X5+1,RECEIVER.BCS,CHAR)
```

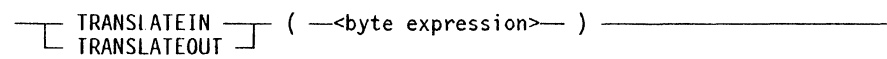
Adaptor Control Translate Functions

The TRANSLATEIN function is used for line-character-to-EBCDIC translation. Normally, this function is used only if RECEIVER.TRANSLATE is FALSE.

The TRANSLATEOUT function is used for EBCDIC-to-line-character translation. Normally, this function is used only if TRANSMITTER.TRANSLATE is FALSE.

Syntax

<adaptor translate function>

 (—<byte expression>—)

Example

```
LASTCHAR := TRANSLATEIN(LASTCHAR)
```

WAITFORIDLE Function

The WAITFORIDLE function delays process execution until one of the following conditions occurs:

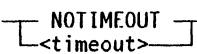
- RECEIVER.INHIBIT becomes TRUE (the ERRORDETECT condition).
- The receiver places a character in the receiver-holding register, indicating the start of a frame (the FRAMEDETECT condition).
- RECEIVER.IDLEDETECT becomes TRUE (the IDLEDETECT condition).
- The number of milliseconds (modulo 32768) specified by < timeout > elapses (the TIMEOUT condition).

A value of type WAITRESULTTYPE is returned indicating the condition that occurred.

If NOTIMEOUT is specified, a timeout value of 0 is used, implying an indefinite wait. The WAITFORIDLE function is allowed only in bit algorithms.

Syntax

<waitforidle function>

— WAITFORIDLE — () —————|

<timeout>

—<integer expression>—————|

Adaptor Control Fault Termination

When an adaptor control process causes an irrecoverable error, the process is terminated, all line control processes for that line are terminated, and all locks that were locked by any of those processes are unlocked. In addition, a result is returned to the host system specifying the line on which the error occurred and the nature of the fault. The host system must reinitialize the line for line activity to resume.

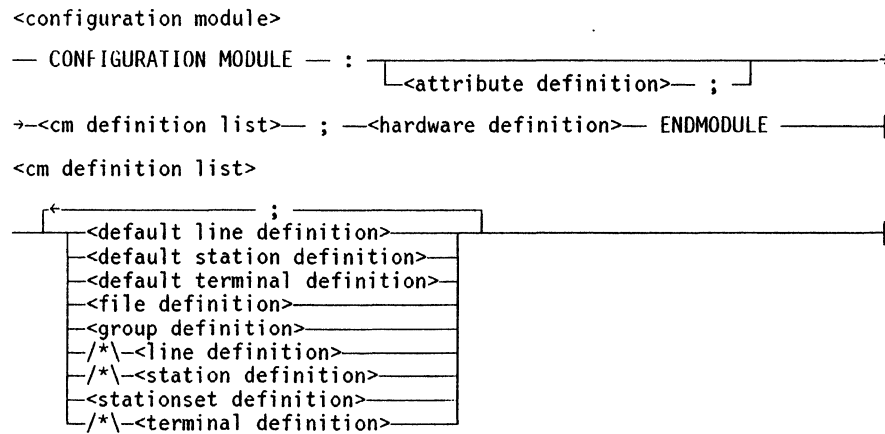
When an adaptor control process executes an ABORT statement, the process is terminated, and the CATEGORY variable of the control block that was INPROGRESS is assigned the value ABORTED. If a control block is queued, its CATEGORY is assigned the value CANCELLED.

Section 4

Configuration Module

The configuration module is structured as a series of definitions. These definitions describe the structure of the data comm network and define the physical and logical attributes of the lines and devices. Definitions are described in the order that they appear in the syntax diagram for the configuration module; attributes within the definitions are presented alphabetically.

Syntax



Explanation

The definitions within the configuration module consist of an identifying keyword (for example, `LINE` for a `LINE` definition), a definition identifier, and a list of attributes and values to be associated with that definition identifier. A definition identifier can be any identifier that is not a reserved word in the configuration module and has not appeared previously as a definition identifier.

Some identifiers declared in the protocol module are visible in the configuration module. These identifiers include algorithm IDs, class IDs, editor IDs, enumerated types, enumerated constants, and variables declared in `INCLUDE` declarations.

ATTRIBUTE Definition

The optional ATTRIBUTE definition allows the definition of additional attributes for a terminal, line, or station. These additional attributes are maintained in the NIFII for use by the host system and are not visible in the protocol module.

Syntax

<attribute definition>

```

— ATTRIBUTE — : — { — /1\—<line attribute declaration> — ; —
                  | — /1\—<station attribute declaration> —
                  | — /1\—<terminal attribute declaration> —

```

<line attribute declaration>

```

— LINE — ( —<attribute body>— ) —————

```

<station attribute declaration>

```

— STATION — ( —<attribute body>— ) —————

```

<terminal attribute declaration>

```

— TERMINAL — ( —<attribute body>— ) —————

```

<attribute body>

```

— { —<Boolean attribute> — ; —
    | —<byte attribute> —
    | —<enumerated attribute> —
    | —<integer attribute> —
    | —<string attribute> —

```

<Boolean attribute>

```

— BOOLEAN —<identifier>— REQUIRED —————
                  | = —<constant Boolean expression>—

```

<byte attribute>

```

— BYTE —<identifier>— REQUIRED —————
                  | = —<constant byte expression>—

```

<enumerated attribute>

```

—<enumerated type>—<identifier>— REQUIRED —————
                  | = —<enumerated constant>—

```

<integer attribute>

```

— INTEGER —<identifier>— REQUIRED —————
                  | = —<constant integer expression>—

```

<string attribute>

```

— STRING —<identifier>— [ —<max size>— ] —————
→ | REQUIRED —————
  | —<constant string expression>—

```

<max size>

```

—<constant integer expression>—

```

Explanation

If an attribute is defined as REQUIRED, that attribute must be given an initial value in the TERMINAL definition, STATION definition, or LINE definition (whichever is appropriate).

The enumerated type in an enumerated attribute must be an enumeration declared globally in the protocol module.

If an initial value is specified for an attribute, this value becomes the value of the attribute unless the attribute is explicitly assigned a different value in the TERMINAL definition, STATION definition, or LINE definition. Enumerated constants must be of the same enumerated type as the variable to which they are assigned.

Example

```
ATTRIBUTE:
  TERMINAL (BOOLEAN SPO REQUIRED);
  STATION  (STRING MCS [17] REQUIRED;
            BYTE PRIORITY = 10);
  LINE     (BOOLEAN DIALONINIT = FALSE)
```

Default Definitions

Default definitions are common attributes that are predefined and recalled for later use.

DEFAULT LINE Definition

The optional DEFAULT LINE definition allows an identifier to be associated with a default set of line attributes. The identifier can then be used in a line DEFAULT specification in a later LINE definition. A default line identifier must appear as the default line ID in a DEFAULT LINE definition before it is used in a line DEFAULT specification.

Syntax

```
<default line definition>
— DEFAULT — LINE —<default line ID>— : —<line attribute list>—|
<default line ID>
—<identifier>—————|
```

DEFAULT STATION Definition

The optional DEFAULT STATION definition allows an identifier to be associated with a default set of station attributes. The identifier can then be used in a station DEFAULT specification in a later STATION definition. A default station identifier must appear as the default station ID in a DEFAULT STATION definition before it can be used in a station DEFAULT specification.

Syntax

```
<default station definition>
— DEFAULT — STATION —<default station ID>— : —————→
→<station attribute list>—————|
<default station ID>
—<identifier>—————|
```

DEFAULT TERMINAL Definition

The optional DEFAULT TERMINAL definition allows an identifier to be associated with a default set of terminal attributes. The identifier can then be used in a terminal DEFAULT spec in a later TERMINAL definition. A default terminal identifier must appear as the default terminal ID in a DEFAULT TERMINAL definition before it can be used in a terminal DEFAULT specification.

Syntax

<default terminal definition>

— DEFAULT — TERMINAL —<default terminal ID>— : —————→

→<terminal attribute list>—————|

<default terminal ID>

—<identifier>—————|

FILE Definition

The optional FILE definition provides a way to define data comm files and to specify the stations associated with those files.

Syntax

```

<file definition>
— FILE —<file ID>— : —<file attribute list>—————|
<file ID>
—<identifier>—————|
<file attribute list>
— { ————— ; ————— }—————|
  — /1\—<files attribute>—————|
  — /1\—<stations attribute>—————|
  — /1\—<title attribute>—————|
<files attribute>
— FILES — = — { <file ID> }—————|
<stations attribute>
— STATIONS — = — { <station ID> }—————|
                  — ALL STATIONS —
<title attribute>
— TITLE — = —<constant string expression>—————|
  
```

Explanation

If ALL STATIONS is specified in the STATIONS attribute, then all stations are members of the file. A single station file need not be defined in a FILE definition, because NDLII assumes that a station ID is a file ID.

The TITLE attribute specifies the title of the file to be used by a host system. If this attribute is not present, the file ID is used as the title. Each title must be unique relative to all other titles that are specified explicitly or implicitly for a STATION definition or FILE definition.

GROUP Definition

The optional GROUP definition allows a line to be associated with a group definition.

Syntax

```

<group definition>
— GROUP —<group ID>— : —<group attribute list>—————|
<group ID>
—<identifier>—————|
<group attribute list>
— { —————|
  /1*\—<group algorithm attribute>—————|
  <group initialize attribute>—————|
  /1*\—<group line ID>—————|

```

Group ALGORITHM Attribute

The required group ALGORITHM attribute names the algorithm in the protocol module that is used by the lines in the group. The line ALGORITHM attribute for each line in the group must specify the same algorithm as does the group ALGORITHM attribute.

Syntax

```

<group algorithm attribute>
— ALGORITHM — = —<algorithm ID>—————|

```

Group INITIALIZE Attribute

The group INITIALIZE attribute allows variables declared for a GROUP structure to be initialized to specified values. The specified variables must have been declared in the variable declaration list of an INCLUDE declaration in the line control part of the algorithm specified by the group ALGORITHM attribute. The enumerated variable ID must be of the same type as the enumerated constant.

Syntax

```

<group initialize attribute>
— INITIALIZE —————→
— { —————|
  <Boolean ID> —:=— <constant Boolean expression>—————|
  <byte ID> —:=— <constant byte expression>—————|
  <enumerated variable ID> —:=— <enumerated constant>—————|
  <integer ID> —:=— <constant integer expression>—————|
  <string ID> —:=— <constant string expression>—————|

```

Group LINE ID Attribute

The required group LINE ID attribute identifies the line that is associated with a group. A particular line can be included in only one group LINE ID. The algorithm specified by the line ALGORITHM attribute of the line in the group LINE ID must be the same as the algorithm specified by the group ALGORITHM attribute.

Syntax

<group line ID attribute>

— LINES — = —<line ID>—————|

section is declared in the algorithm. In either case, the line CLASS attribute of the line must be compatible with the selected adaptor control. When the line is declared in a group LINE ID, the algorithm specified for the line must be the same as that specified for the group in the group ALGORITHM attribute.

Syntax

<line algorithm attribute>

— ALGORITHM — = —<algorithm ID> — . —<adaptor control ID> —

Line CLASS Attribute

The required line CLASS attribute must reference the class ID of a line class declared in the protocol module. The class IDs of all terminals referenced by all stations in the line STATION list for the line must be identical to the class ID of the line.

Syntax

<line class attribute>

— CLASS — = —<class ID> —

Line DISCONNECTACTION Attribute

The optional line DISCONNECTACTION attribute assigns the initial state of the LINE.DISCONNECTACTION variable. This variable determines the method to be used by the executive to connect a disconnected switched line. If this attribute is not specified, the default value for DISCONNECTACTION is NONE. The value of this attribute is ignored for nonswitched lines.

Syntax

<line disconnectaction attribute>

— DISCONNECTACTION — = —
— AUTOANSWER —
— LINEDIAL —
— MANUOLDIAL —
— NONE —
— STNSETDIAL —
— STNSETDIALANSWER —

Line DPRDELAY Attribute

The optional line DPRDELAY attribute specifies the number of milliseconds (modulo 32768) of delay during autodial between the time a dial digit is loaded on signals NB1 through NB8 for the ACU and the time the digit present (DPR) signal is raised. If this attribute is not specified, it has a default value of 0, which is the normal value for telephone equipment used in the United States.

Syntax

<line dprdelay attribute>
 — DPRDELAY — = —<constant integer expression>—————|

Line FUNCTION Attribute

The line FUNCTION attribute specifies the function of the host system on a bit-mode line. This function affects the operation of the adaptor control RECEIVE statement and TRANSMIT statement.

Syntax

<line function attribute>
 — FUNCTION — = —| PRIMARY —————|
 | SECONDARY —<select address>|
 <select address>
 —<constant byte expression>—————|

Explanation

For the *FUNCTION = SECONDARY* form, the select address specifies the address used by the receiver to select which incoming frames are to be accepted.

This attribute is required for bit-mode lines and cannot be specified for either asynchronous-mode or synchronous-mode lines.

Line INITIALIZE Attribute

The optional line INITIALIZE attribute allows variables declared for a LINE structure to be initialized to specified values. The specified variables must have been declared in the variable declaration list of an INCLUDE declaration in the line control part of the algorithm specified by the line ALGORITHM attribute. The enumerated variable ID must be of the same type as the enumerated constant.

Syntax

<line initialize attribute>
 — INITIALIZE —————→
 → {
 <Boolean ID> — := —<constant Boolean expression>———|
 <byte ID> — := —<constant byte expression>—————|
 <enumerated variable ID> — := —<enumerated constant>———|
 <integer ID> — := —<constant integer expression>—————|
 <string ID> — := —<constant string expression>—————|
 }

Explanation

The line INITIALIZE attribute must not appear before the line has been associated with an algorithm either explicitly or by invocation of a line DEFAULT specification. A line

ALGORITHM attribute following a line INITIALIZE attribute must specify the same algorithm as the line ALGORITHM attribute in the invoked line DEFAULT specification (the algorithm on which the initialization was based).

Line LOSSOFCARRIER Attribute

The optional line LOSSOFCARRIER attribute specifies the action to be taken when a loss of carrier occurs on a switched line.

Syntax

<line lossofcarrier attribute>

— LOSSOFCARRIER — = — DISCONNECT —
 NODISCONNECT

Explanation

If DISCONNECT is specified, the line is automatically disconnected. If NODISCONNECT is specified or if the attribute is not specified, the line is not automatically disconnected.

Normally, the DISCONNECT option is used with switched modems that operate with a continuous carrier. The NODISCONNECT option must be used on a modem that uses a controlled carrier.

Line PHONE Attribute

The optional line PHONE attribute specifies the dial sequence used to dial out on the line. This attribute must be specified if the line is to be eligible for line autodial. (Refer to the discussion of the predeclared LINE structure variable DISCONNECTACTION under “Predeclared LINE Structure Variables” in Section 3.)

Syntax

<line phone attribute>

— PHONE — = —<dial sequence>—

<dial sequence>

—<dial digit>—

<dial digit>

<digit>
*
#
E
W
D1
D2
D3
D4
D5
D6
D7
D8
D9

Explanation

The character E denotes the end-of-number digit. The character W indicates a wait for a supplementary dial tone. The D_n sequence indicates a preliminary or interdigit delay, where *n* is in the range of from 1 to 9 seconds.

Line READY Attribute

The optional line READY attribute specifies an initial value for the LINE.READY variable. This variable is used to control the suspension of line control processes by the host. If this attribute is not specified, it has a default value of TRUE.

Syntax

<line ready attribute>

— READY — = ☐ TRUE ☐ FALSE

Line RECEIVEDLAY Attribute

The optional line RECEIVEDLAY attribute specifies the time, in milliseconds, that the line waits after an adaptor control ENABLE RECEIVER statement before the receiver receives characters from the line. RECEIVEDLAY can assume values from 0 to 32767. If this attribute is not specified, a RECEIVEDLAY of 0 is assumed.

When you assign a value to this attribute, consider the value of the terminal RECEIVEDLAY attribute. If there is a difference between the line RECEIVEDLAY attribute value for a given line and the terminal RECEIVEDLAY attribute value for one or more terminals on the line, then the NDLLI compiler uses the line RECEIVEDLAY attribute value.

Syntax

<line receivedelay attribute>

— RECEIVEDLAY — = —<constant integer expression>—

Line SETDIGITDELAY Attribute

The optional line SETDIGITDELAY attribute specifies the number of milliseconds (modulo 32768) of delay during autodial between the time when the ACU unit raises the present next digit (PND) signal and the time the next dial digit is loaded on signals NB1 through NB8. If this attribute is not specified, it has a default value of 0, which is the normal value for telephone equipment used in the United States.

Syntax

<line setdigitdelay attribute>

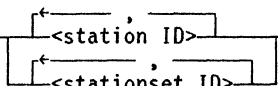
— SETDIGITDELAY — = —<constant integer expression>—————|

Line STATION List

The line STATION list enumerates the stations that are associated with the line. If the line is included in a group LINE ID in a GROUP definition, this attribute must not appear in the LINE definition. If the line is not part of a group, the line STATION list must be included in the LINE definition.

Syntax

<line station list>

— STATIONS — = ——————|

Explanation

The *STATIONS* = <station ID>, . . . form of the line STATION list requires the specification of at least one station ID and is used for both point-to-point contention configurations and nonswitched polled configurations. If a particular station is included in a stationset member list, it cannot be included in a line STATION list; otherwise, it can be included in at most one line STATION list.

The algorithm specified by the station ALGORITHM attribute of each station in the line STATION list must be the same as the algorithm specified by the line ALGORITHM attribute.

The *STATIONS* = <stationset ID>, . . . form of the line STATION list requires the specification of at least one stationset ID and is used for switched polled configurations. A particular stationset can be included in at most one line STATION list.

The algorithm specified by the station ALGORITHM attribute of each station of each stationset in the line STATION list must be the same as the algorithm specified by the line ALGORITHM attribute.

Line TIMEOUT Attribute

The optional line TIMEOUT attribute specifies the default timeout value used in the adaptor control RECEIVE statement. If a character is not received within the milliseconds (modulo 32768) specified by the value of the constant integer expression, a TIMEOUT error is reported. If this attribute is not specified, it has a default value of 0 (no time limit).

Syntax

<line timeout attribute>

— TIMEOUT — = —<constant integer expression>—————|

Line TRANSMITDELAY Attribute

The optional line TRANSMITDELAY attribute specifies the time, in milliseconds, that the line waits after an adaptor control ENABLE TRANSMITTER statement before the transmitter sends characters onto the line.

Syntax

<line transmitdelay attribute>

— TRANSMITDELAY — = —<constant integer expression>—————|

Explanation

TRANSMITDELAY can assume values from 0 to 32767. If this attribute is not specified, a TRANSMITDELAY of 0 is assumed.

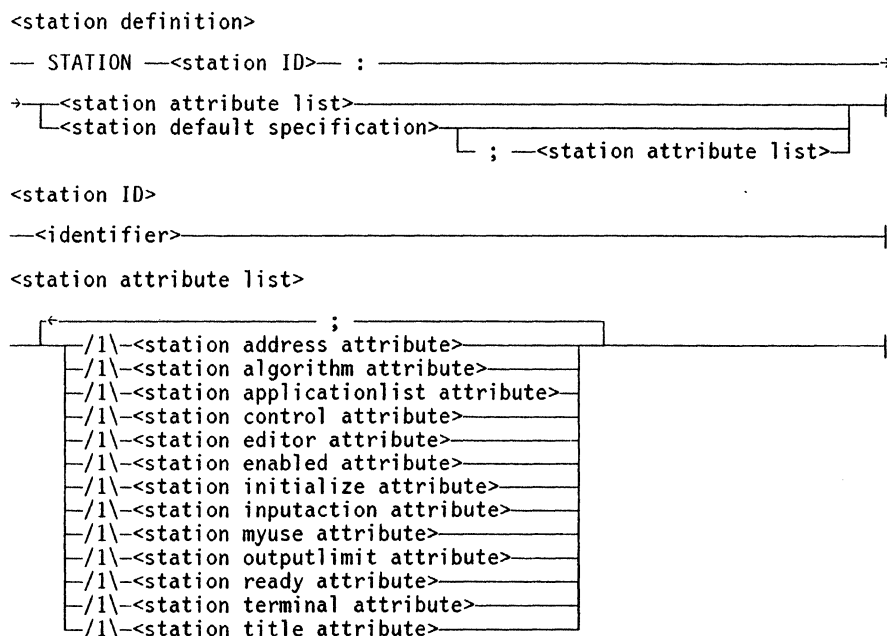
When you assign a value to this attribute, consider the value of the terminal TRANSMITDELAY attribute. If there is a difference between the line TRANSMITDELAY attribute value for a given line and the terminal TRANSMITDELAY attribute value for one or more terminals on the line, then the NDLLI compiler uses the line TRANSMITDELAY attribute value.

STATION Definition

The required STATION definition declares the attributes of stations in the data comm subsystem. The required and optional components of the STATION definition are defined and discussed in the following subsections.

The maximum number of stations allowed by the NDLLI compiler is 7000.

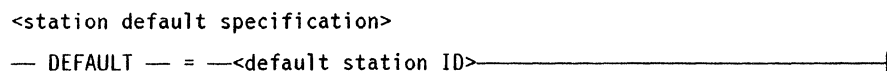
Syntax



Station DEFAULT Specification

The optional station DEFAULT specification invokes a previously defined DEFAULT STATION definition, which is specified by its default station ID. Any station attributes specified in the selected DEFAULT STATION definition are used as default attribute specifications for this STATION definition, as if they had been defined at this point. If any of the default attributes are also explicitly specified in the STATION definition, the explicitly defined attribute values override those specified in the DEFAULT STATION definition.

Syntax



Station ADDRESS Attribute

The station ADDRESS attribute defines the address characters for the terminal of the station. This attribute is required if the terminal ADDRESSLENGTH attribute specifies

Line SETDIGITDELAY Attribute

The optional line SETDIGITDELAY attribute specifies the number of milliseconds (modulo 32768) of delay during autodial between the time when the ACU unit raises the present next digit (PND) signal and the time the next dial digit is loaded on signals NB1 through NB8. If this attribute is not specified, it has a default value of 0, which is the normal value for telephone equipment used in the United States.

Syntax

<line setdigitdelay attribute>

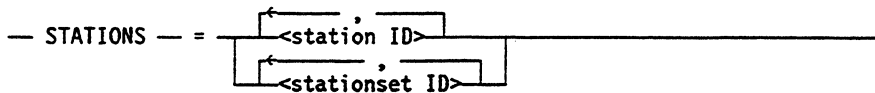
— SETDIGITDELAY — = —<constant integer expression>—————|

Line STATION List

The line STATION list enumerates the stations that are associated with the line. If the line is included in a group LINE ID in a GROUP definition, this attribute must not appear in the LINE definition. If the line is not part of the GROUP definition, inclusion of the line STATION list is optional; however, Unisys recommends that you include the line STATION list. Failure to include a line STATION list makes a line unusable unless you modify the DATACOMINFO file with the Interactive Datacomm Configurator (IDC).

Syntax

<line station list>



Explanation

The *STATIONS = <station ID>, . . .* form of the line STATION list requires the specification of at least one station ID and is used for both point-to-point contention configurations and nonswitched polled configurations. If a particular station is included in a stationset member list, it cannot be included in a line STATION list; otherwise, it can be included in at most one line STATION list.

The algorithm specified by the station ALGORITHM attribute of each station in the line STATION list must be the same as the algorithm specified by the line ALGORITHM attribute.

The *STATIONS = <stationset ID>, . . .* form of the line STATION list requires the specification of at least one stationset ID and is used for switched polled configurations. A particular stationset can be included in at most one line STATION list.

The algorithm specified by the station ALGORITHM attribute of each station of each stationset in the line STATION list must be the same as the algorithm specified by the line ALGORITHM attribute.

Syntax

<station enabled attribute>

— ENABLED — =

TRUE
FALSE

Station INITIALIZE Attribute

The optional station INITIALIZE attribute allows variables declared for a STATION structure to be initialized to specified values. The specified variables must have been declared in the variable declaration list of an INCLUDE declaration in the line control part of the algorithm specified by the station ALGORITHM attribute. The enumerated variable ID must be of the same type as the enumerated constant.

Syntax

<station initialize attribute>

— INITIALIZE —

<Boolean ID> := <constant Boolean expression>
<byte ID> := <constant byte expression>
<enumerated variable ID> := <enumerated constant>
<integer ID> := <constant integer expression>
<string ID> := <constant string expression>

Explanation

The station INITIALIZE attribute cannot appear before the station has been associated with an algorithm either explicitly or by invocation of a station DEFAULT specification. If a station declaration contains both a station DEFAULT specification and a station ALGORITHM attribute and the algorithm for the default station is not the same as the algorithm for the station being declared, then a station INITIALIZE attribute must be declared for the station. If you do not explicitly declare a station INITIALIZE attribute for the station, the station INITIALIZE attribute variables for the default station are used, and these variables might not be compatible with the algorithm for the station being declared.

Station INPUTACTION Attribute

The optional station INPUTACTION attribute assigns the initial state of the STATION.INPUTACTION variable. This variable determines the action performed by the executive when an input result is returned for the station. If this attribute is not specified, INPUTACTION has a default value of SEND.

Syntax

<station inputaction attribute>

— INPUTACTION — =

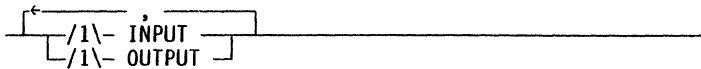
SEND
SENDANDWAIT

Station MYUSE Attribute

The optional station MYUSE attribute specifies that a station is input only, output only, or both input and output. If this attribute is not specified, both input and output are assumed.

Syntax

<station myuse attribute>

— MYUSE — = 

Explanation

The value of this attribute is stored in the NIFII for use by the host system, as follows:

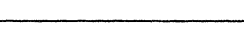
- If MYUSE is OUTPUT, the host system allows a remote file to be opened on this station for output only. The host system does not enable the station for input.
- If MYUSE is INPUT, the host system allows a remote file to be opened on this station for input only. The host system does not send an output request to the station.
- If both INPUT and OUTPUT are specified, the host system allows remote files to be opened on this station for input, output, or both.

Station OUTPUTLIMIT Attribute

The optional station OUTPUTLIMIT attribute specifies the maximum number of output messages that will be accepted by the NSP to be queued for the station.

Syntax

<station outputlimit attribute>

— OUTPUTLIMIT — = —<constant integer expression>— 

Explanation

The constant integer expression value ranges from 1 to 255 with a default value of 7.

Queued output messages for all the stations connected to an NSP are stored in the NSP memory. Use of the station OUTPUTLIMIT attribute in large networks can reduce NSP memory requirements.

Station READY Attribute

The optional station READY attribute assigns the initial state of the STATION.READY variable. This variable determines whether or not a line control process can assign its station reference variable to reference this station through execution of a

NEXTSTATION statement. If this attribute is not specified, READY has a default value of TRUE.

Syntax

<station ready attribute>

— READY — =

TRUE
FALSE

Station TERMINAL Attribute

The required station TERMINAL attribute specifies the type of terminal associated with the station. The terminal ID must have appeared previously in a TERMINAL definition.

Syntax

<station terminal attribute>

— TERMINAL — = —<terminal ID>_____

Station TITLE Attribute

The optional station TITLE attribute specifies the title of a station to be used by a host system. If this attribute is not present, the station ID is used as the title. Each title must be unique relative to all other titles specified either explicitly or implicitly for a STATION definition or FILE definition.

Syntax

<station title attribute>

— TITLE — = —<constant string expression>_____

STATIONSET Definition

The optional STATIONSET definition allows the grouping of stations by a remote site so that either a single switched line or a group of switched lines can easily control stations at a number of remote sites. The members of a stationset must all be jointly physically accessible (whenever the stationset is connected to the subsystem, all members of the stationset must be accessible).

When a line STATION list is composed of stationsets, the executive maintains a list of associations between the line and stationsets. When a line and a stationset are associated with each other, that line can access only the members of that stationset (using a NEXTSTATION statement) and the members of that stationset can be accessed only by that line. When a line is not associated with a stationset, that line can access only the members of stationsets that are also unassociated.

The executive forms an association between a line and stationset whenever either of the following occurs: the line is disconnected (thus, not associated with a stationset) and the executive performs a successful autodial on that line for the stationset; or the line is connected but not associated with a stationset and performs a successful NEXTSTATION statement with a select part. The executive breaks an association between a line and stationset whenever the line becomes disconnected.

The LINE.SYSTEM and LINE.STATUSCHANGED variables respond to line-stationset associations. If a line and stationset are associated, the values of these variables for that line are affected only by the status of stations that are members of that stationset. If a line is not associated with a stationset, the values of these variables for that line are affected only by the status of stations that are members of unassociated stationsets.

Syntax

<stationset definition>

— STATIONSET —<stationset ID>— : —<stationset attribute>—

<stationset ID>

—<identifier>—

<stationset attribute>

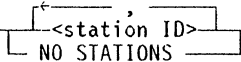
—/1*\-<stationset member list>—
—/1*\-<stationset phone attribute>—

Stationset Member List

The required stationset member list enumerates the stations that are members of the stationset. A particular station can be included in at most one stationset member list, and a station included in a stationset member list cannot be included in a line STATION list.

Syntax

<stationset member list>

— STATIONS — = —  —

Explanation

If the NO STATIONS option is used, a stationset with a null station list is created. Such a stationset can be referenced in a LINE definition in order to define a line that initially has no stations assigned to it.

Stationset PHONE Attribute

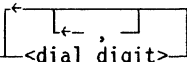
The optional stationset PHONE attribute specifies the dial sequence used to dial out to the remote site that corresponds to the stationset. This attribute must be specified in order for the stationset to be eligible for stationset autodial. (Refer to the discussion of the predeclared LINE structure variable DISCONNECTACTION under “Predeclared LINE Structure Variables” in Section 3.)

Syntax

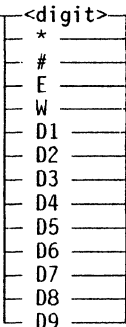
<stationset phone attribute>

— PHONE — = — <dial sequence> —

<dial sequence>

 —

<dial digit>

 —

Explanation

The character E denotes the end-of-number digit. The character W indicates a wait for a supplementary dial tone. The Dn sequence indicates a preliminary or interdigit delay, where n is in the range of from 1 to 9 seconds.

TERMINAL Definition

Syntax

```

<terminal definition>
— TERMINAL —<terminal ID>— : —————→
→ <terminal attribute list>—————|
  | <terminal default specification> |
  | ; —<terminal attribute list>— |

<terminal ID>
—<identifier>—————|

<terminal attribute list>
←————— ; —————|
  | /1\<terminal addresslength attribute>—————|
  | /1\<terminal class attribute>—————|
  | /1\<terminal linewidth attribute>—————|
  | /1\<terminal maxinput attribute>—————|
  | /1\<terminal maxoutput attribute>—————|
  | /1\<terminal pagecount attribute>—————|
  | /1\<terminal pagesize attribute>—————|
  | /1\<terminal receivedelay attribute>—————|
  | /1\<terminal screen attribute>—————|
  | /1\<terminal transmitdelay attribute>—————|
  | /1\<terminal type attribute>—————|
  | /1\<terminal wraparound attribute>—————|

```

Terminal DEFAULT Specification

The optional terminal DEFAULT specification invokes a previously defined DEFAULT TERMINAL definition, which is specified by its default terminal ID. Any terminal attributes that are specified in the selected DEFAULT TERMINAL definition are used as default attribute specifications for the TERMINAL definition in which the terminal DEFAULT specification appears, as if these attributes had been defined at this point. If any of the default attributes are also explicitly specified in the TERMINAL definition, the explicit attribute specifications override the values assigned in the DEFAULT TERMINAL definition.

Syntax

```

<terminal default specification>
— DEFAULT — = —<default terminal ID>—————|

```

Terminal ADDRESSLENGTH Attribute

The optional terminal ADDRESSLENGTH attribute defines the number of address characters that the terminal transmits and receives. If the ADDRESSLENGTH attribute is not assigned a value, the remote device is assumed not to use device addressing, and ADDRESSLENGTH has a default value of 0.

Syntax

```

<terminal addresslength attribute>
— ADDRESSLENGTH — = —————→
→ [ /1*\- RECEIVE — : —<receive address size> ]
  [ /1*\- TRANSMIT — : —<transmit address size> ]
<receive address size>
—<constant integer expression>—————|
<transmit address size>
—<constant integer expression>—————|

```

Explanation

The receive address size and transmit address size define the number of characters in the receive and transmit addresses, respectively, and must be in the range 0 to 3.

The address sizes of all terminals referenced by stations in the same line STATION list must be identical.

Terminal CLASS Attribute

The required terminal CLASS attribute references the name of a class declared in a CLASS declaration in the protocol module. The class IDs of all terminals referenced by stations in the same line STATION list must be identical.

Syntax

```

<terminal class attribute>
— CLASS — = —<class ID>—————|

```

Terminal LINEWIDTH Attribute

The optional terminal LINEWIDTH attribute defines the number of graphic characters that can be displayed or printed on one terminal line.

Syntax

```

<terminal linewidth attribute>
— LINEWIDTH — = —<constant integer expression>—————|

```

Explanation

If the LINEWIDTH attribute is assigned 0 or allowed to assume its default value of 0, the width of the line is unspecified.

Terminal MAXINPUT Attribute

The required terminal MAXINPUT attribute specifies the maximum number of bytes that can be received from a terminal in one transmission.

Syntax

<terminal maxinput attribute>

— MAXINPUT — = —<constant integer expression>—————|

Terminal MAXOUTPUT Attribute

The optional terminal MAXOUTPUT attribute specifies the maximum number of bytes that can be transmitted to a terminal in one transmission. If the MAXOUTPUT attribute is not assigned a value, it has a default value equal to the value of the MAXINPUT attribute.

Syntax

<terminal maxoutput attribute>

— MAXOUTPUT — = —<constant integer expression>—————|

Terminal PAGECOUNT Attribute

The optional terminal PAGECOUNT attribute specifies the number of pages of data that the terminal can buffer. If the PAGECOUNT attribute is not assigned a value, it has a default value of 0.

Syntax

<terminal pagecount attribute>

— PAGECOUNT — = —<constant integer expression>—————|

Terminal PAGESIZE Attribute

The optional terminal PAGESIZE attribute defines the maximum number of output lines that can be sent before requiring special control information. A value of 0 indicates that no special page control action is required. If the PAGESIZE attribute is not assigned a value, it has a default value of 0.

Syntax

<terminal pagesize attribute>

— PAGESIZE — = —<constant integer expression>—————|

Terminal RECEIVEDelay Attribute

The optional terminal RECEIVEDelay attribute is used for certain data terminal equipment (DTE) and modems that require a minimum time period to stabilize before transmitting. The RECEIVEDelay attribute introduces a delay before any characters are received to avoid the detection of errors during the stabilization period.

Syntax

<terminal receivedelay attribute>

— RECEIVEDelay — = —<constant integer expression>—————|

Explanation

The value for the RECEIVEDelay attribute is in milliseconds and must be in the range 0 to 32767. The default value for the RECEIVEDelay attribute is 0.

When you assign a value to this attribute, consider the value of the line RECEIVEDelay attribute. If there is a difference between the line RECEIVEDelay attribute value for a given line and the terminal RECEIVEDelay attribute value for one or more terminals on the line, then the NDLLI compiler uses the line RECEIVEDelay attribute value.

Terminal SCREEN Attribute

The optional terminal SCREEN attribute specifies whether the terminal is a CRT device (TRUE) or hardcopy device (FALSE). If not specified, the SCREEN attribute has a default value of FALSE.

Syntax

<terminal screen attribute>

— SCREEN — = —

TRUE
FALSE

 —————|

Terminal TRANSMITDelay Attribute

The optional terminal TRANSMITDelay attribute is used for certain data terminal equipment (DTE) and modems that require a minimum time period to stabilize before receiving. The TRANSMITDelay attribute introduces a delay before any characters are transmitted to avoid the loss of data during the stabilization period.

Syntax

<terminal transmitdelay attribute>

— TRANSMITDelay — = —<constant integer expression>—————|

Explanation

The value for the TRANSMITDELAY attribute is in milliseconds and must be in the range 0 to 32767. The default value for the TRANSMITDELAY attribute is 0.

When you assign a value to this attribute, consider the value of the line TRANSMITDELAY attribute. If there is a difference between the line TRANSMITDELAY attribute value for a given line and the terminal TRANSMITDELAY attribute value for one or more terminals on the line, then the NDLLI compiler uses the line TRANSMITDELAY attribute value.

Terminal TYPE Attribute

Each terminal must have an associated TYPE attribute declared in the protocol module TERMINALTYPE declaration. An enumerated constant of the enumerated type TERMINALTYPE is maintained in the STATION structure variable TYPE.

Syntax

<terminal type attribute>

— TYPE — = —<enumerated constant>—————|

Terminal WRAPAROUND Attribute

When TRUE, the terminal WRAPAROUND attribute causes a terminal to perform automatic line feed and carriage return when displaying a character in the last character position of a line. The output editor should check this attribute to avoid double-spacing the output. If not assigned a value, the WRAPAROUND attribute has a default value of TRUE.

Syntax

<terminal wraparound attribute>

— WRAPAROUND — =

TRUE
FALSE

—————|

HARDWARE DEFINITION

The required **HARDWARE DEFINITION** defines the network support processors (NSPs), the line support processors (LSPs) associated with the NSPs, and the connections of lines to line adaptors in the data comm subsystem.

Syntax

<hardware definition>

— **HARDWARE DEFINITION** — : —<hardware module>—

<hardware module>

— **HOST** —<host NSP ID> : —<option list>— ; —

→ —<LSP module>—

<host NSP ID>

—<constant integer expression>—

<NSP ID>

—<constant integer expression>—

<option list>

— **EDITOR** — = —<editor ID>—

<LSP module>

— **LSP** —<LSP ID> : —
 —/1\—<LSP alternates>
 —/1\—<LSP mode>—

→ —/16\—<LSP adaptor declaration>—

<LSP ID>

—<constant integer expression>—

<LSP alternates>

— **ALTERNATES** — = —
 NSP —<NSP ID>
 HOST —<host NSP ID>— ; —

<LSP mode>

— **MODE** — = —
 STANDARD
 NATIVE — ; —

<LSP adaptor declaration>

— **ADAPTOR** — <LSP adaptor number> — : —<line ID>—

<LSP adaptor number>

—<constant integer expression>—

Explanation

The basic data comm subsystem hardware is composed of two data link processors (DLPs), the LSP DLP and the NSP DLP. A data comm subsystem can also contain data communications data link processors (DCDLPs), enhanced data communications data link processors (EDCDLPs), or data communications adapters (DCAs). A DCDLP or DCA must be declared as an NSP with exactly one LSP, and the LSP can have up to four adaptors, numbered 0 through 3. The NSP ID must be the physical unit number of the DCDLP; the LSP ID is ignored by the MCP and need only be unique. An EDCDLP must be declared in the same manner as a DCDLP, except that the LSP can have up to eight adaptors, numbered 0 through 7.

The adaptor control portion of an algorithm in the protocol module runs in the LSP.

LSPs can be connected to either an NSP or the host system. In configurations that include an NSP or DCDLP, line control and the editors run in the NSP or DCDLP. When the LSPs are connected directly to the host system, line control and the editors run in the host system.

The host NSP IDs and NSP IDs are integers in the range 0 to 4095 that uniquely identify hosts and NSPs. Each host NSP ID and NSP ID must be unique relative to all other host NSP IDs and NSP IDs.

All editors associated with stations assigned to the specified hardware module are automatically loaded when the data comm subsystem is initialized. The EDITOR option specifies additional editors that can be loaded into the specified hardware module.

An LSP module must appear for each LSP used by the data comm subsystem. The LSP ID must be in the range 0 to 4095 and must be unique for all LSPs specified in the HARDWARE DEFINITION. In addition, the LSP ID must be different from all host NSP IDs and NSP IDs specified.

An LSP alternates definition can be specified to give a list of other NSPs or host configurations that have visibility of that LSP and that the host system can use if reconfiguration of the LSP is necessary. The ordering of NSPs or host configurations in the LSP alternates list specifies the order in which the host should attempt to find another NSP or host configuration to control that LSP.

An LSP mode definition can be specified to select standard or native firmware only when external adaptor control is not specified in any of the line IDs.

An LSP adaptor declaration must be specified for each adaptor on an LSP that is used by the data comm subsystem. The LSP adaptor number is an integer representing the adaptor location of the line within the LSP. LSP adaptor numbers must be unique for each LSP. LSP adaptor numbers must be in the range 0 to 15 for physical NSPs, 0 to 3 for DCDLPs or DCAs, and 0 to 7 for EDCDLPs. The line ID must be the name of a line (specified in the LINE definition) that is to be associated with the specified adaptor location on the specified LSP. A particular line ID can be referenced at most once within the HARDWARE DEFINITION.

If a line ID uses an adaptor control that is declared to be external, all adaptors on the LSP must use the same adaptor control. However, adaptors located on a DCDLP are not physically associated with an LSP; the adaptors on DCDLPs, therefore, can use different adaptor controls.

Configuration Module Example

This example demonstrates how to write a configuration module. It is not intended to cover all combinations or conditions that might be required, or to fit any one existing configuration.

CONFIGURATION MODULE:

```
ATTRIBUTE: LINE      (BOOLEAN ENDOFNUMBER = FALSE;  
                     INTEGER MAXSTATIONS = 10);  
            STATION  (BYTE PRIORITY = 10);
```

DEFAULT TERMINAL TD830TERMDEF:

```
ADDRESSLENGTH  = RECEIVE: 2, TRANSMIT: 2;  
CLASS          = TDCLASS;  
LINEWIDTH      = 40;
```

TERMINAL TD830TERM:

```
DEFAULT        = TD830TERMDEF;  
LINEWIDTH      = 80;  
MAXINPUT       = 1920;  
MAXOUTPUT      = 1920;  
PAGECOUNT     = 2;  
PAGESIZE       = 24;  
RECEIVEDELAY   = 600;  
SCREEN         = TRUE;  
TRANSMITDELAY  = 750;  
TYPE           = TD830TYPE;  
WRAPAROUND     = TRUE;
```

DEFAULT TERMINAL TD830TERMDEF1:

```
CLASS          = MYCLASS;  
MAXINPUT       = 1920;  
MAXOUTPUT      = 1920;  
ADDRESSLENGTH  = RECEIVE: 2, TRANSMIT: 2;
```

TERMINAL TD830TERM1:

```
DEFAULT        = TD830TERMDEF1;  
PAGECOUNT     = 1;  
PAGESIZE       = 33;  
RECEIVEDELAY   = 555;  
TRANSMITDELAY  = 777;  
LINEWIDTH      = 72;  
WRAPAROUND     = FALSE;  
TYPE           = TD830TYPE;  
SCREEN         = TRUE;
```

```

DEFAULT STATION STATIONDEF1:
    EDITOR          = NONE;
    TERMINAL        = TD830TERM;
    ALGORITHM       = TD;
    APPLICATIONLIST = CANDE_EDITOR: CANDE_EDITOR;

STATION TD830STA:
    DEFAULT         = STATIONDEF1;
    MYUSE           = INPUT, OUTPUT;
    ADDRESS         = RECEIVE: "X1", TRANSMIT: "X1";
    INPUTACTION     = SENDANDWAIT;

STATION TD830STA2:
    DEFAULT         = STATIONDEF1;
    ALGORITHM       = TD;
    ADDRESS         = RECEIVE: "X2", TRANSMIT: "X2";
    MYUSE           = OUTPUT;
    TERMINAL        = TD830TERM;

STATION TD830STA3:
    ALGORITHM       = TD;
    TERMINAL        = TD830TERM;
    ADDRESS         = RECEIVE: "X2", TRANSMIT: "X2";

STATION TD830STA4:
    ADDRESS         = RECEIVE: "X2", TRANSMIT: "X2";
    ALGORITHM       = TD;
    TERMINAL        = TD830TERM;

STATION TD830STA5:
    ALGORITHM       = TD;
    TERMINAL        = TD830TERM;
    ADDRESS         = RECEIVE: "X2", TRANSMIT: "X2";

STATION TD830STA6:
    ALGORITHM       = TD;
    TERMINAL        = TD830TERM;
    ADDRESS         = RECEIVE: "X2", TRANSMIT: "X2";

STATION TD830STA7:
    ALGORITHM       = TD;
    TERMINAL        = TD830TERM;
    ADDRESS         = RECEIVE: "X2", TRANSMIT: "X2";
    PRIORITY        = 20;

```



```
DEFAULT STATION STATIONDEF2:
    EDITOR          = CANDE_EDITOR;
    TERMINAL        = TD830TERM1;
    ALGORITHM       = TD;

STATION TD830STA13:
    DEFAULT         = STATIONDEF2;
    ALGORITHM       = TD;
    ADDRESS         = RECEIVE: "X2", TRANSMIT: "X2";

STATIONSET STATIONSETA:
    STATIONS        = TD830STA4, TD830STA5;

DEFAULT LINE TDLINE:
    ALGORITHM       = TD.TDAC;      % CHARACTER
    CLASS          = TDCLASS;      % MODE=ASYNC (CHAR)

LINE TDLINE1:
    DEFAULT         = TDLINE;
    TIMEOUT        = 59;
    DPRDELAY       = 69;
    PHONE          = D9;
    READY          = TRUE;
    RECEIVEDDELAY  = 79;
    SETDIGITDELAY  = 89;
    TRANSMITDELAY  = 99;
    MAXSTATIONS    = 20;
    ENDOFNUMBER    = TRUE;
    DISCONNECTACTION
                  = LINEDIAL;

LINE TDLINE2:
    DEFAULT         = TDLINE;
    ALGORITHM       = TD.TDAC;
    CLASS          = TDCLASS;
    TRANSMITDELAY  = 1500;

LINE TDLINE3:
    DEFAULT         = TDLINE;
    ALGORITHM       = TD.TDAC;
    CLASS          = TDCLASS;
    RECEIVEDDELAY  = 1200;

LINE TDLINE4:
    ALGORITHM       = TD.TDAC;
    CLASS          = TDCLASS;
    STATIONS       = TD830STA3;

LINE TDLINE5:
    ALGORITHM       = TD.TDAC;
    CLASS          = TDCLASS;
    STATIONS       = TD830STA6;
```

```

LINE TDLIN6:
    ALGORITHM      = TD.TDAC;
    CLASS          = TDCLASS;
    STATIONS       = TD830STA7;
    PHONE          = 33;

```

```

GROUP GP1:
    ALGORITHM      = TD;
    LINES          = TDLIN1;

```

```

GROUP GP2:
    ALGORITHM      = TD;
    LINES          = TDLIN3;

```

HARDWARE DEFINITION:

```

NSP 69: EDITOR = CANDE_EDITOR;
LSP 55:
    ADAPTOR 1: TDLIN1;    % TD830 "XX" BY ODT
    ADAPTOR 2: TDLIN2;
    ADAPTOR 3: TDLIN5;

```

```

NSP 70: EDITOR = CANDE_EDITOR;
LSP 66:
    ALTERNATES = NSP 71;
    %MODE      = NATIVE;
    ADAPTOR 1: TDLIN4;

```

```

NSP 71: EDITOR = CANDE_EDITOR;
LSP 77:
    ALTERNATES = NSP 70;
    %MODE      = STANDARD;
    ADAPTOR 1: TDLIN6;

```

```

ENDMODULE

```


HARDWARE DEFINITION

The required **HARDWARE DEFINITION** defines the network support processors (NSPs), the line support processors (LSPs) associated with the NSPs, and the connections of lines to line adaptors in the data comm subsystem.

Syntax

<hardware definition>

— **HARDWARE DEFINITION** — : —<hardware module>—

<hardware module>

— **HOST** —<host NSP ID> : —<option list>— ;

→ —<LSP module>—

<host NSP ID>

—<constant integer expression>—

<NSP ID>

—<constant integer expression>—

<option list>

— **EDITOR** — = —<editor ID>—

<LSP module>

— **LSP** —<LSP ID> : —
—/1\—<LSP alternates>
—/1\—<LSP mode>—

→ —/16\—<LSP adaptor declaration>—

<LSP ID>

—<constant integer expression>—

<LSP alternates>

— **ALTERNATES** — = —
— **NSP** —<NSP ID> ;
— **HOST** —<host NSP ID>—

<LSP mode>

— **MODE** — = —
— **STANDARD** — ;
— **NATIVE** —

<LSP adaptor declaration>

— **ADAPTOR** —<LSP adaptor number> — : —<line ID>—

<LSP adaptor number>

—<constant integer expression>—

Explanation

The basic data comm subsystem hardware is composed of two data link processors (DLPs), the LSP DLP and the NSP DLP. A data comm subsystem can also contain data communications data link processors (DCDLPs), enhanced data communications data link processors (EDCDLPs), or data communications host adapters (DCHAs). A DCDLP or DCHA must be declared as an NSP with exactly one LSP, and the LSP can have up to four adaptors, numbered 0 through 3. The NSP ID must be the physical unit number of the DCDLP; the LSP ID is ignored by the MCP and need only be unique. An EDCDLP must be declared in the same manner as a DCDLP, except that the LSP can have up to eight adaptors, numbered 0 through 7.

The adaptor control portion of an algorithm in the protocol module runs in the LSP.

LSPs can be connected to either an NSP or the host system. In configurations that include an NSP or DCDLP, line control and the editors run in the NSP or DCDLP. When the LSPs are connected directly to the host system, line control and the editors run in the host system.

The host NSP IDs and NSP IDs are integers in the range 0 to 4095 that uniquely identify hosts and NSPs. Each host NSP ID and NSP ID must be unique relative to all other host NSP IDs and NSP IDs.

All editors associated with stations assigned to the specified hardware module are automatically loaded when the data comm subsystem is initialized. The EDITOR option specifies additional editors that can be loaded into the specified hardware module.

An LSP module must appear for each LSP used by the data comm subsystem. The LSP ID must be in the range 0 to 4095 and must be unique for all LSPs specified in the HARDWARE DEFINITION. In addition, the LSP ID must be different from all host NSP IDs and NSP IDs specified.

An LSP alternates definition can be specified to give a list of other NSPs or host configurations that have visibility of that LSP and that the host system can use if reconfiguration of the LSP is necessary. The ordering of NSPs or host configurations in the LSP alternates list specifies the order in which the host should attempt to find another NSP or host configuration to control that LSP.

An LSP mode definition can be specified to select standard or native firmware only when external adaptor control is not specified in any of the line IDs.

An LSP adaptor declaration must be specified for each adaptor on an LSP that is used by the data comm subsystem. The LSP adaptor number is an integer representing the adaptor location of the line within the LSP. LSP adaptor numbers must be unique for each LSP. LSP adaptor numbers must be in the range 0 to 15 for physical NSPs, 0 to 3 for DCDLPs or DCHAs, and 0 to 7 for EDCDLPs. The line ID must be the name of a line (specified in the LINE definition) that is to be associated with the specified adaptor location on the specified LSP. A particular line ID can be referenced at most once within the HARDWARE DEFINITION.

Appendix A

System Messages

This appendix lists in alphabetical order the messages generated by NDLL run-time errors that cause fault-termination of processes. Each message is presented as it appears to the operator, followed by a description of the error conditions that generated the message.

DIVIDE BY ZERO

Cause: The DIV modulus in a DIV function, or the MOD modulus in a MOD function, was zero.

EMPTY SOURCE

Cause: One of the following line control statements failed as described:

- A DEALLOCATE statement was executed when <CB variable>.TEXT was NULL.
- A DISCARDRESULT statement was executed when STATION.RESULT was NULL.
- An ENQUEUE statement was executed when STATION.REQUEST or STATION.RESULT was NULL.
- A FINISHREQUEST statement was executed when the station reference variable or STATION.REQUEST was NULL.
- A MOVETEXT statement was executed in one of the following situations:
 - For the <request-to-CB> form, when STATION.REQUEST.TEXT was NULL.
 - For the <CB-to-request> and <CB-to-result> forms, when <CB variable>.TEXT was NULL.
 - For the <CB-to-CB> form, when <source CB variable>.TEXT was NULL.
- A NEXTREQUEST statement was executed when STATION.QUEUED was FALSE.
- A PURGE statement was executed when no request in STATION.REQUESTLIST, or no result in STATION.RESULTLIST, was associated with the specified key.

- A REBLOCK statement was executed when `<CB variable>.TEXT` was NULL.
- A SENDHOST statement was executed when `STATION.RESULT` was NULL.

FETCH OVERFLOW

Cause: An adaptor control FETCH statement was executed when `ENDTEXT` was TRUE.

INVALID ADDRESS

Cause: One of the following error conditions occurred:

- A control block array was accessed using a CB subscript greater than the CB limit specified in the CB declaration for the array.
- A CB declaration, EVENT declaration, or INTERLOCK declaration was incompletely specified in an external line control declaration.

INVALID CASE VALUE

Cause: A case statement did not specify a case guard for which `<case selector> = <case guard>` and did not specify an *ELSE:* guard.

INVALID KERNEL CALL

Cause: A line control UNLOCK statement specified a lock that was not locked or was locked by another process.

INVALID OPERAND

Cause: One of the following error conditions occurred:

- An integer expression specified either a field width that was not in the range 0 to < field start > or a < field start > that was not in the range 1 to 16 when the field width was not 0.
- An integer function either contained a string expression with a non-EBCDIC digit or resulted in a value that was greater than 65535.
- A STRINGADD function or STRINGSUBTRACT function contained a string expression with a non-EBCDIC digit.
- An attempt was made to reference one of the STATION.REQUEST or STATION.RESULT structure variables when STATION.REQUEST or STATION.RESULT was NULL.
- STATION was used as a qualifier when the station reference variable was NULL.
- A control block array was accessed using a CB subscript with a value less than 1.
- < CB variable > .CATEGORY was INPROGRESS when one of the following line control forms was executed:
 - ALLOCATE statement
 - COPYSTRING statement
 - COPYSTRUCTURE statement
 - DEALLOCATE statement
 - INITIATE statement
 - REBLOCK statement
 - MOVETEXT statement (< request-to-CB > , < CB-to-request > , and < CB-to-result > forms)
- A line control ENQUEUE statement specified a key with a value greater than 255.
- A line control INITIATE statement initiated a special operation when < CB variable > .TEXT was not NULL.
- A MOVETEXT statement (< request-to-CB > form) was executed when STATION.REQUEST was NULL.
- A MOVETEXT statement (< CB-to-result > form) was executed when STATION.RESULT was NULL.
- A MOVETEXT statement (< CB-to-CB > form) was executed when the source CB variable was NULL or when the destination CB variable was NULL.

- A Line Control REBLOCK statement specified a RESIDUE IN that had a value greater than or equal to the original blocking factor.
- A line control TIMEINTERVAL function specified one or more integers in the TIMESTAMP string that were not in the range of values specified for the predeclared line control variable TIMESTAMP.

NULL STATION

- Cause:* One of the following line control statements was executed when the station reference variable was NULL:
- CLEAR STATION statement
 - DEQUEUE statement
 - DISCARDRESULT statement
 - ENQUEUE statement
 - NEWSRESULT statement
 - NEXTREQUEST statement
 - PURGE statement
 - SENDHOST statement
 - MOVETEXT statement (<request-to-CB>, <CB-to-request>, or <CB-to-result> form)

OCCUPIED DESTINATION

- Cause:* One of the follow conditions occurred:
- One of the following line control statements was executed when <CB variable>.TEXT was not NULL:
 - ALLOCATE statement
 - COPYSTRING statement
 - COPYSTRUCTURE statement
 - MOVETEXT statement (<request-to-CB> form)
 - A DEQUEUE statement or NEXTREQUEST statement was executed when STATION.REQUEST was not NULL.
 - A DEQUEUE statement or NEWSRESULT statement was executed when STATION.RESULT was not NULL.

- An ENQUEUE statement was executed specifying a key that was already associated with a request in the STATION.REQUESTLIST or with a result in the STATION.RESULTLIST.
- A MOVETEXT statement of the <CB-to-request> form was executed when STATION.REQUEST.TEXT was not NULL.
- A MOVETEXT statement of the <CB-to-result> form was executed when STATION.RESULT.TEXT was not NULL.
- A MOVETEXT statement of the <CB-to-CB> form was executed when <destination CB variable>.TEXT was not NULL.

PROTECTED NAME

Cause: An attempt was made to enter data in a Read-Only or NULL structure.

RECEIVER ALREADY ENABLED

Cause: An adaptor control ENABLE statement was executed to enable a receiver when RECEIVER.ENABLED was TRUE.

RECEIVER NOT ENABLED

Cause: An adaptor control DISABEL statement or RECEIVE statement was executed for a receiver when RECEIVER.ENABLED was FALSE.

STRING PROTECT

Cause: One of the following string operations failed as described:

- A DROP function specified a drop count whose value exceeded the length of the specified string expression.
- A STRING function either specified a result length greater than 255 or resulted in a value that exceeded the specified result length.
- A STRINGADD function either specified an add result length greater than 255 or resulted in a value that was greater than 255 when an asterisk (*) was specified as the add result length.

- A SUBSTRING function specified one of the following:
 - A substring start outside the range 1 to the length of the specified string expression
 - A substring start and substring length whose sum minus 1 was greater than the length of the specified string expression
- A TAKE function specified a take count greater than the number of characters in the specified string expression.
- A string subscript used to extract a single byte value from a string variable had a value less than 1 or greater than the current length of the string.
- The length of the string resulting from a concatenation operation was greater than 255.
- The length of a complex string expression stored in a string variable exceeded the declared string max size for the string variable.

STRUCTURE PROTECT

Cause: The structure subscript used to access a structure within a structure array had a value greater than the declared structure size or equal to 0.

TRANSMITTER ALREADY ENABLED

Cause: An adaptor control ENABLE statement was executed to enable a transmitter when TRANSMITTER.ENABLED was TRUE.

TRANSMITTER NOT ENABLED

Cause: An adaptor control DISABLE statement or TRANSMIT statement was executed for a transmitter when TRANSMITTER.ENABLED was FALSE.

Appendix B

NDLII Compiler and NDLII Post Compiler

This appendix describes the files used by the NDLII compiler and the compiler control records accepted by the compiler. Compiler control records contain options that control the input and output of the compiler. The appendix also describes how to use the NDLII post compiler (NPC).

NDLII Compiler

You use the NDLII compiler to create a network information file II (NIFII). The following pages describe the files used by the compiler, and the control records that instruct the compiler how to process the input files.

Compiler Files

The files used by the NDLII compiler are illustrated in Figure B-1.

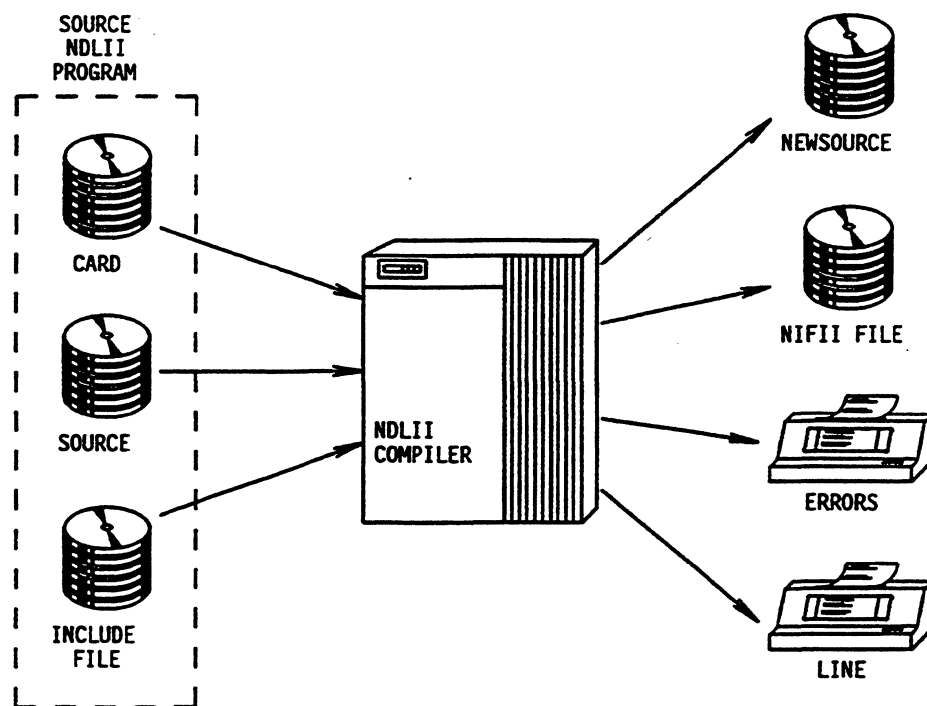


Figure B-1. Compiler Files

The attributes of the NDLII compiler files are briefly described in the following table. A more detailed description of each compiler file follows the table.

Internal Name/Default Title	Default Device	Purpose
CARD	READER	Primary input
SOURCE	DISK	Secondary input
NEWSOURCE	DISK	Source output
LINE	PRINTER	Printer output (listing)
CODE	DISK	Object code output (NIFII)
ERRORS	PRINTER/ REMOTE	Error output

CARD is the first file that the compiler reads. If the MERGE option is disabled, CARD contains the entire source language input of the program to be compiled. If the MERGE option is enabled, CARD contains controls directing the compiler to a stored source file that is patched by source input from the CARD file.

SOURCE is the file from which the compiler reads previously stored source records. The compiler is directed to this file if the MERGE option is enabled. Patches to this file can appear in the CARD file.

NEWSOURCE is the output source file of the compiler. This source file is either a copy of the CARD input (if the MERGE option is disabled) or the result of merging the CARD file and the SOURCE file (if the MERGE option is enabled). This file is suitable for use as a source file for a subsequent compilation and is created only if the NEW option is enabled.

LINE is the file the compiler generates as an output listing, if any.

CODE contains the NIFII generated from the NDLII program. The NIFII contains the following kinds of information:

- Code for NDLII-declared processes (editors, line controls, and adaptor controls)
- Data structures for lines, stations, and so forth
- Physical hardware configuration of the network
- Logical attributes of lines, stations, and so forth
- Global host/NDLII definitions

ERRORS is the file to which the compiler writes error messages when the ERRORLIST option is enabled. The default device for this file is PRINTER except for interactive compilations, for which the default device is REMOTE.

Compiler Control Records

A compiler control record contains a dollar sign (\$) in column 1 of the input record. Records that begin with a single dollar sign in column 1 affect only the current compilation; these records, called temporary compiler control records, are not saved in

[illegible]

<errorlimit option>	
<pagesize option>	
<sequence base option>	
<sequence increment option>	
<title option>	
<trainid option>	
<version option>	

<code option>
<delete option>
<errorlist option>
<lineinfo option>
<list option>
<list\$ option>
<listdeleted option>
<listincl option>
<listp option>
<map option>
<merge option>
<new option>
<seqcheck option>
<sequence option>
<summary option>
<upcaselisting option>
<warnfatal option>
<xref option>

```

    <clear option>
    <include option>
    <page option>
    <void option>

```

An immediate option causes the compiler to perform a function independent of subsequent processing.

The keywords SET, RESET, and POP affect the values of Boolean options. Each Boolean option has an associated stack in which up to 47 previous values of the option are saved. The management of the option value and the stack is as follows:

- If a Boolean option is not preceded anywhere in the compiler control record by a SET, RESET, or POP keyword, it is implicitly enabled, and the previous value is pushed onto the stack (that is, the previous value of the option is saved).
- If a Boolean option appears in a list of one or more options following a SET or RESET keyword, the option is enabled or disabled, respectively, and the previous value is pushed onto the stack.
- If a Boolean option appears in a list of one or more options following a POP keyword, the current value of the option is discarded, and the value on top of the stack is popped off the stack to become the current value of the option (the option is restored to its last previous value).
- If the CLEAR option appears, all Boolean options except the MERGE option and the NEW option are disabled, and all Boolean option stacks are initialized to all FALSE.

Compiler Control Options

Compiler control options include the following:

CLEAR	LISTINCL	Sequence increment
CODE	LISTP	SUMMARY
DELETE	MAP	TITLE
ERRORLIMIT	MERGE	TRAINID
ERRORLIST	NEW	UPCASELISTING
INCLUDE	PAGE	VERSION
LINEINFO	PAGESIZE	VOID
LIST	SEQCHECK	WARNFATAL
LIST\$	SEQUENCE	XREF
LISTDELETED	Sequence base	

CLEAR Option

(Type: Immediate)

The CLEAR option causes the compiler to disable all BOOLEAN options except the MERGE option and, conditionally, the NEW option. If a source language record has been written to the output source file (NEWSOURCE) because the NEW option is enabled, the NEW option is not disabled.

Syntax

<clear option>

— CLEAR —————|

This option can appear only on a compiler control record in the primary input file (CARD).

CODE Option

(Type: Boolean, Default: FALSE)

When enabled, the CODE option causes the compiler to list the object code produced by the compilation.

Syntax

<code option>

— CODE —————|

DELETE Option

(Type: Boolean, Default: FALSE)

When enabled, the DELETE option causes the compiler to discard source language records from the secondary input file (SOURCE) until the option is disabled. The value of the DELETE option is ignored if the MERGE option has not been enabled.

Syntax

<delete option>

— DELETE —————|

Explanation

The source language records discarded as a result of this option are not carried forward to the output source file (NEWSOURCE), and they are not listed unless the LISTDELETED option is enabled.

The DELETE option can appear only on a compiler control record in the primary input file (CARD).

ERRORLIMIT Option

(Type: Value, Default: 100)

The ERRORLIMIT option specifies the maximum number of errors that the compiler can produce before the compilation is terminated.

Syntax

<errorlimit option>

```
— ERRORLIMIT            <integer> _____
```

The integer must be an unsigned integer in the range 0 to 549,755,813,887. If the error limit is exceeded, the compiler produces a listing of the errors, informs the user that the compilation was terminated because the error limit was exceeded, and purges the NEWSOURCE file (if any). The compiler keeps a count only of syntax errors and not of warnings, unless the WARNFATAL option is enabled.

ERRORLIST Option

(Type: Boolean, Default: TRUE for interactive compilations, FALSE otherwise)

When enabled, the **ERRORLIST** option causes compiler diagnostic messages to be written to a separate output error file (**ERRORS**).

Syntax

<errorlist option>

— ERRORLIST —

If the **ERRORLIST** option is enabled and the **LIST** option is disabled, compiler diagnostic messages appear only in the output error file (**ERRORS**). If both the **ERRORLIST** option and the **LIST** option are enabled, compiler diagnostic messages appear in both the output error file (**ERRORS**) and output listing (**LINE**).

INCLUDE Option

(Type: Immediate)

The **INCLUDE** option causes the compiler to suspend input from the primary and secondary input files (CARD and SOURCE) and to accept input from the file specified by file title.

Syntax

```
<include option>
```

```
— INCLUDE — " —<file title>— " —<starting sequence number>—
```

→ T0 —<ending sequence number>

<starting sequence number>

—<integer>—

<ending sequence number>

—<integer>—

Explanation

The file title must conform to the rules for file titles as defined in the *A Series Work Flow Language (WFL) Programming Reference Manual*.

If a starting sequence number is specified, the inclusion of the source records begins with the first record in the included file that has a sequence number greater than or equal to the starting sequence number. If a starting sequence number is not specified, inclusion begins with the first record of the included file. If an ending sequence number is specified, the inclusion of the source records ends when a record in the included file has a sequence number greater than the ending sequence number. If an ending sequence number is not specified, the inclusion of source records ends following the last record in the included file.

The starting sequence number and ending sequence number must be in the range 0 to 99999999, inclusive.

A file that is included can itself contain compiler control records with INCLUDE options. INCLUDE options can be nested to a maximum of five levels.

The source language records included as a result of the INCLUDE option are not listed unless the LISTINCL option is enabled, and they are not carried forward to the output source file (NEWSOURCE).

No compiler control options can follow the INCLUDE option on a given compiler control record.

LINEINFO Option

(Type: Boolean, Default: FALSE)

When enabled, the LINEINFO option causes information associating source record sequence numbers with object code addresses to be stored in the NIFII. This information is useful for determining the point of program termination if the program terminates abnormally.

Syntax

<lineinfo option>

— LINEINFO —————|

LIST Option

(Type: Boolean, Default: FALSE for interactive compilations, TRUE otherwise)

When enabled, the LIST option causes the compiler to list the source language input, including the NDLII statements and permanent compiler control records. In addition, the information printed when the LISTP option and SUMMARY option are enabled is printed.

Syntax

<list option>

— LIST —————|

LIST\$ Option

(Type: Boolean, Default: FALSE)

When enabled, the LIST\$ option causes the compiler to list all temporary compiler control records encountered during the compilation.

Syntax

<list\$ option>

— LIST\$ —————|

LISTDELETED Option

(Type: Boolean, Default: FALSE)

When enabled, the LISTDELETED option causes the compiler to list all source language input that was deleted because the DELETE option or VOID option was enabled.

Syntax

<listdeleted option>

— LISTDELETED —————|

LISTINCL Option

(Type: Boolean, Default: FALSE)

When enabled, the LISTINCL option causes the compiler to list all source language input that was accepted for compilation as a result of an INCLUDE option.

Syntax

<listincl option>

— LISTINCL —————|

LISTP Option

(Type: Boolean, Default: FALSE)

When enabled, the LISTP option causes the compiler to list source language records that originated from the primary input file (CARD). If the LIST option is enabled, the LISTP option has no effect.

Syntax

<listp option>

— LISTP —————|

MAP Option

(Type: Boolean, Default: FALSE)

When enabled, the MAP option causes the compiler to include, as part of the output listing, information concerning the allocation of variables within the object code produced by the compilation.

Syntax

<map option>

— MAP —————|

MERGE Option

(Type: Boolean, Default: FALSE)

When enabled, the MERGE option causes the compiler to merge source language records from the primary input file (CARD) with source language records from the secondary input file (SOURCE). The file title, if present, causes the specified file to be read as the SOURCE file. The file title must conform to the rules for file titles as defined in the *A Series Work Flow Language (WFL) Programming Reference Manual*. If no file title is specified, SOURCE is assumed.

Syntax

<merge option>

— MERGE —

[" —<file title>— "] —————|

Explanation

Once enabled, this option remains enabled throughout a compilation and cannot be disabled. Subsequent attempts to enable or disable this option are treated as errors and are ignored.

This option can appear only on a compiler control record in the primary input file (CARD).

NEW Option

(Type: Boolean, Default: FALSE)

When enabled, the NEW option causes the compiler to create a new source language output file (NEWSOURCE) consisting of all source language records accepted for compilation (not including those discarded by enabling the DELETE option or VOID option).

Syntax

<new option>

— NEW — " —<file title>— " —

Explanation

The file title, if present, causes the specified title to be assigned to the NEWSOURCE file. The file title must conform to the rules for file titles as defined in the *A Series Work Flow Language (WFL) Programming Reference Manual*. If no file title is specified, NEWSOURCE is assumed.

Once enabled, the NEW option remains enabled throughout a compilation and cannot be disabled. Subsequent attempts to enable or disable this option are treated as errors and are ignored.

This option can appear only on a compiler control record in the primary input file (CARD).

PAGE Option

(Type: Immediate)

The PAGE option causes the compiler to eject a page on the output listing. If the LIST option is disabled, the PAGE option is ignored.

Syntax

<page option>

— PAGE —

PAGESIZE Option

(Type: Value, Default: 58)

The PAGESIZE option causes the compiler to store a value indicating the maximum number of lines that can appear on a page of the output listing. The integer must be an unsigned integer in the range 3 to 549,755,813,887. The page size specified includes the number of lines necessary for the page heading.

Syntax

<pagesize option>

— PAGESIZE —<integer>—————|

SEQCHECK Option

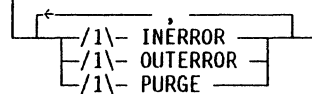
(Type: Boolean, Default: FALSE)

When enabled, the SEQCHECK option causes the compiler to verify that the sequence number of the current source language record acquired from the primary input file (CARD) or the secondary input file (SOURCE), or directed to the output source file (NEWSOURCE), is greater than the sequence number of the previous source language record acquired from or directed to the same file.

Syntax

<seqcheck option>

— SEQCHECK —

**Explanation**

If the sequence numbers are not in ascending order, a warning message is produced. For the primary input file only, if the previous sequence number was all spaces and the current sequence number is also all spaces, a warning message is not produced.

INERROR causes the compiler to treat these warnings on the primary or secondary input file as syntax errors. OUTERROR causes the compiler to treat these warnings on the output source file (NEWSOURCE) as syntax errors. PURGE causes the compiler to purge the output source file (NEWSOURCE) if sequence errors occur in the output source file.

SEQUENCE (SEQ) Option

(Type: Boolean, Default: FALSE)

When enabled, the SEQUENCE option causes the compiler to assign new sequence numbers to the source language records accepted for compilation. SEQUENCE and SEQ are synonymous. This option affects only input source language records from the primary (CARD) or secondary (SOURCE) input files.

Syntax

<sequence option>

— SEQUENCE
— SEQ —————|

Explanation

The SEQUENCE option assigns the current sequence base (see “Sequence Base Option” in this appendix) to the current source language record and increments the sequence base by the sequence increment (see “Sequence Increment Option” in this appendix). Sequencing occurs before the record is written to the output source file (NEWSOURCE) and before the source language record is listed. If the sequence number exceeds 99999999 after the compiler has incremented the sequence base by the sequence increment, the SEQUENCE option is disabled and an error message is produced.

Sequence Base Option

(Type: Value, Default: 1000)

The sequence base option causes the compiler to store the specified integer value as the sequence base associated with the SEQUENCE option. The maximum value of the integer is 99999999. This option can be specified independently of the SEQUENCE option.

Syntax

<sequence base option>

— <integer> ————— |

Sequence Increment Option

(Type: Value, Default: 1000)

The sequence increment option causes the compiler to store the specified integer value as the sequence increment associated with the SEQUENCE option. The maximum value of the integer is 99999999. This option can be specified independently of the SEQUENCE option.

Syntax

<sequence increment option>

— + — <integer> ————— |

SUMMARY Option

(Type: Boolean, Default: FALSE)

When enabled, the SUMMARY option causes the compiler to produce a summary of information about the compilation on the output listing. The summary produced when this option is enabled is the same as the summary produced when the LIST option is enabled. The SUMMARY option takes effect only if the LIST option is disabled.

Syntax

<summary option>

— SUMMARY —————|

TITLE Option

(Type: Value, Default: the title of the CARD file if the MERGE option is disabled; the title of the SOURCE file if the MERGE option is enabled)

The TITLE option causes the compiler to store a title to be used on subsequent pages of the output listing. The title string must contain at least one character and is truncated if it is longer than 29 characters. The quotation mark (") cannot appear within the title string. Each subsequent use of this option causes a new title string to be stored in place of the previous title string.

Syntax

<title option>

— TITLE ————|

<title string>

— " ————|

TRAINID Option

(Type: Value, Default: none)

The TRAINID option causes the compiler to assign the specified value to the TRAINID attribute for all printer files that the compiler produces. The TRAINID option affects the LINE file, the ERRORS file (for non-CANDE compilations), and the printed output of the cross-reference information, if any.

Syntax

<trainid option>

— TRAINID ————|

Explanation

If the TRAINID option is not used, the compiler does not assign a value to the TRAINID attribute for these files.

To have any effect, the TRAINID option must appear before the compiler opens its printer files (for safety, this option should appear on the first input record). If an attempt is made to use the TRAINID option after the LINE file is open, a warning message is generated and the value of the option is not changed.

UPCASELISTING Option

(Type: Boolean, Default: FALSE)

When the UPCASELISTING option is enabled, all output to the LINE file and the ERRORS file is translated to uppercase before it is written.

Syntax

<upcaselisting option>

— UPCASELISTING —————|

VERSION Option

(Type: Value, Default: none)

The VERSION option assigns a version number to be used during the compilation for the patch mark field.

Syntax

<version option>

— VERSION —<version number>—————|

<version number>

—<release number>— . —<cycle number>—|
 | . —<patch number>—|

<release number>

—|←-/1\—|
—|<digit>—|—————|

<cycle number>

—|←-/2\—|
—|<digit>—|—————|

<patch number>

—|←-/2\—|
—|<digit>—|—————|

Explanation

If the patch mark field (columns 81 through 90) of a record contains a valid patch number and a VERSION option has been encountered, the patch mark field is changed to the following form in the NEWSOURCE file and in the LINE file:

<release number>.<cycle number>.<patch number>

The version number specified in the first VERSION option in an NDLII program replaces the version number that appears on any later VERSION option. The version number is updated in the NEWSOURCE file if the NEW option is enabled and in the LINE file if the LIST option is enabled. If the patch number is not present in the new VERSION option, the old patch number is retained.

A VERSION option that is to be updated must be the last item on the compiler control record and must allow sufficient room on the control record for the updated version number to be inserted.

A warning message is given if the new version number is less than the old version number.

VOID Option

(Type: Immediate)

The VOID option causes all input records in the secondary input file (SOURCE) to be discarded until the secondary input sequence number exceeds the specified sequence number.

Syntax

<void option>

— VOID —<sequence number>—————|

<sequence number>

—<integer>—————|

Explanation

The sequence number must be an unsigned integer less than or equal to 99999999. The sequence number cannot be less than the sequence number of the compiler control record on which the VOID option is specified. The VOID option is ignored if the MERGE option is not enabled.

The source language records discarded as a result of the VOID option are not carried forward to the output source file (NEWSOURCE), and they are not listed unless the LISTDELETED option is enabled.

The VOID option can appear only on a compiler control record in the primary input file (CARD).

WARNFATAL Option

(Type: Boolean, Default: FALSE)

When enabled, the WARNFATAL option causes the compiler to treat warning messages as syntax errors.

Syntax

<warnfatal option>

— WARNFATAL —————|

XREF Option

(Type: Boolean, Default: FALSE)

When enabled, the XREF option causes the compiler to generate cross-reference information about the source language program accepted for compilation.

Syntax

<xref option>

— XREF —

NDLII Post Compiler (NPC)

The NDLII post compiler (NPC) reads an NDLII code file (NIFII) and converts the algorithms and editors found there into code that can run on an EDCDLP or a DCHA.

You cannot select the algorithms that are to be post compiled. The NPC post compiles all protocols that fit into its currently accepted requirements.

The NPC currently has the following NDLII restrictions:

- Only one process per line control is permitted.
- Only character adaptor controls are permitted.
- Only the standard CRC16 and $Xn + 1$ polynomials are supported.
- Control block arrays are not permitted.
- EVENT, LOCK, station REQUESTLIST, and station RESULTLIST variables are not permitted.
- The REBLOCK function is not permitted.
- No more than one INITIATE statement can be outstanding per adaptor process, except for special INITIATE statements, such as CANCEL and TESTADAPTER.

Refer to the *A Series Data Communications Protocols Installation and Implementation Guide* for a list of algorithms supported by the NPC.

The NPC generates a listing that indicates which protocols were post compiled and which were not. In the latter case, the listing provides the reason that the protocol was not post compiled. If a particular protocol is skipped, no post compiled code is generated for it.

The main benefit in using the NPC is the ability to use NDLII to develop protocols for the EDCDLP or the DCHA. If you wish to add algorithms or change an existing algorithm, you need only make changes to the NDLII source, compile it, then post compile. The NPC also allows the EDCDLP or DCHA to emulate a network support processor/line support processor (NSP/LSP) on fewer cards at a lower cost. For more information on the EDCDLP and the DCHA, refer to the *A Series Data Communications Protocols Installation and Implementation Guide*.

Figure B-2 illustrates the files used by the NPC.



Figure B-2. NDLII Post Compiler (NPC) Files

The code for EDCDLP protocols can be found in SOURCENDLII or in the file called FIRMWARE/NSP/050/ < editor or line control name > or NPC/ < protocol name > /DATACOMINFO. The code for DCHA protocols can be found in SOURCENDLII or in the file called FIRMWARE/NSP/076/ < editor or line control name > or NPC/ < protocol name > /DATACOMINFO. If you have not purchased the NDLII post compiler, any purchased protocols are either firmware files or DATACOMINFO files. External protocols such as BDLC, SDLC, and BSC 56K BPS are packaged as firmware files. The files that you have purchased are installed from the system tape during the Simple Installation process. You must copy purchased protocols that are packaged as separate DATACOMINFO files (one per protocol) to your current DATACOMINFO file after they have been installed.

When the EDCDLP is initialized, the DCC converts the new NIFII into a DATACOMINFO file, downloads the executive from the file named FIRMWARE/NSP/050/EXECUTIVE, and downloads all the editors and algorithms from the DATACOMINFO file. When the DCHA is initialized, the process is similar except that the file from which the executive is downloaded is called FIRMWARE/NSP/076/EXECUTIVE. If external protocols are declared in the NDLII source, they will also be downloaded by the DCC. Figure B-3 illustrates this process.

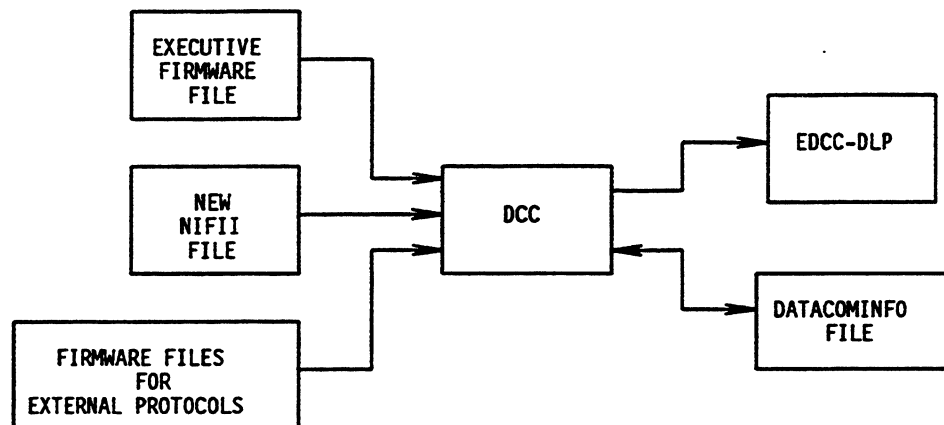


Figure B-3. Initializing EDCDLP or DCHA

Configuration Overview

Use the following instructions to implement EDCDLP or DCHA protocols:

1. Begin with the SOURCENDLII file. Use an editor to modify an algorithm if you have a custom protocol or to modify the source to reflect your site requirements.
2. Run the source file through the NDLII compiler to produce a NIFII. You can do this with a Work Flow Language (WFL) job or with the Command and Edit (CANDE) COMPILE command. Note that the last node of the file name must be NIF because IDC and the ID system command look for NIF as the suffix.

The following is an example of the CANDE *COMPILE* command:

```
COMPILE SOURCENDLII AS <file name prefix>/NIF WITH NDLII
```

3. If you purchased the NDLII post compiler (NPC), run the NIFII through the NPC to produce a new NIFII that will contain code for the EDCDLP or DCHA. Then run the IDC to produce the DATACOMINFO file.

Refer to “Running the Post Compiler” later in this appendix for details about executing the NPC.

If you did not purchase the NPC, run the IDC to produce the DATACOMINFO file. Then use the IDC to copy the appropriate algorithm and editor for each unbundled protocol into the current DATACOMINFO file.

The following is an example of the IDC *COPY* command for copying the algorithm and editor for the HASP protocol:

```
COPY ALGORITHM HASP FROM NPC/HASP  
COPY EDITOR HASPED FROM NPC/HASP
```

4. Designate the new DATACOMINFO file by using the system command *ID* <file name prefix>. The variable <file name prefix> is the prefix of a file name of a DATACOMINFO file or a NIFII.

The following is an example of the ID system command:

```
ID CURRENT
```

For additional information, refer to the ID system command in the *A Series System Commands Operations Reference Manual*.

5. To initialize datacomm, enter the ID system command in the following form:

```
ID <DLP number>
```

Running the Post Compiler

You can run the NPC by using either a WFL job deck or the CANDE *COMPILE* command.

Using a WFL Job Deck

The following examples show how to build up a WFL job deck to post compile a NIFII.

Example 1

```
COMPILE NEW/NIF WITH NPC TO LIBRARY;  
COMPILER FILE HOST = OLD/NIF;
```

To run the NPC, you must use the WFL statement `COMPILE`. You must also equate an internal file named `HOST` to the NIFII that is to be post compiled (as shown in the second line of this example).

Example 2

```
COMPILE NEW/NIF WITH NPC TO LIBRARY;  
COMPILER FILE HOST = OLD/NIF;  
COMPILER DATA CARD  
$ SET LIST CODE  
? END
```

Use a `CARD` input file to designate options for the NPC as shown in lines 3 and 4 in Example 2. Valid options are listed in "CARD File User Options" later in this appendix.

Example 3

```
COMPILE NEW/NIF WITH NPC TO LIBRARY;  
COMPILER FILE HOST = OLD/NIF;  
COMPILER FILE SOURCE = SYMBOL/SOURCENDLII;  
COMPILER DATA CARD  
$ SET LIST CODE  
? END
```

You can equate the `SOURCE` input file to the original NDLII source that was used to create the NIFII. The output `CODE` listing will show NDLII source statements if the `LINEINFO` option was `SET` in the original version of that source and if the `CODE` option is `SET` in the `CARD` file of this job deck.

Using the CANDE *COMPILE* Command

You can also run the NPC with `CANDE`. The following steps provide an example of this process:

1. Create a file name by using the `MAKE` command. For example:

```
MAKE NPC/CARD
```

2. Put compiler control records into the file that you created. For example:

```
$ SET LIST CODE  
$ RESET ERRORLIST
```

3. Save the file that you created. For example:

SA

4. Use the *CANDE COMPILE* command to compile the program. For example:

```
C NPC/CARD AS $NEW/NIF WITH NPC;  
C FILE HOST = OLD/NIF;  
C FILE SOURCE = SYMBOL/SOURCENDLII
```

Card File User Options

Table B-1 shows user options that can be designated in the CARD file for the NPC. The records in the CARD file must begin with a dollar sign (\$) in column 1 followed by either SET or RESET and the option names.

Table B-1. CARD File User Options

Option	WFL Default	CANDE Default	Use
LIST	TRUE	FALSE	Creates a listing that contains the names of all algorithms and editors as they are encountered and a message that indicates whether or not the algorithms and editors were post compiled. The listing also contains the reason why a particular editor or algorithm was not post compiled.
CODE	FALSE	FALSE	Lists the 8086 code generated for all protocols being post compiled.
ERRORLIST	FALSE	TRUE	Causes diagnostic messages to be written to a separate output error file.
WARNFATAL	FALSE	FALSE	Causes the compilation to terminate abnormally if any warnings are encountered.
WARNSUPR	FALSE	FALSE	Does not print any warning message. If WARNFATAL is TRUE, the program ignores this option.
UPCASELISTING	FALSE	FALSE	Formats output to print in uppercase letters if this option is TRUE.

Appendix C

Reserved and Recognized Words

This appendix contains reserved words and recognized words from the protocol module and reserved words from the configuration module.

Reserved Words in the Protocol Module

The following list contains the reserved words for the protocol module:

ABORT	ABORTTYPE	ADAPTOR
ALGORITHM	ALLOCATE	APPLICATIONTYPE
BEGIN	BIT	BLOCK
BOOLEAN	BYTE	CALL
CANCEL	CASE	CAUSE
CBCATEGORY	CHARACTER	CLASS
CLEAR	CONFIGURATION	CONTROL
COPYSTRING	COPYSTRUCTURE	DEALLOCATE
DEFINE	DEQUEUE	DISABLE
DISCARDRESULT	DISCONNECTPOLICY	DO
DUPLICATE	EDITOR	ELSE
ENABLE	END	ENDMODULE
ENDPROCEDURE	ENDPROCESS	ENQUEUE
ENUMERATION	EVENT	FETCH
FINISHREQUEST	IF	INCLUDE
INITIALIZE	INITIATE	INPUTPOLICY
INTEGER	INTERLOCK	LIST
LOAD	LOADSTRUCTURE	LOCK
MODULE	MOVETEXT	NEWRESULT
NEXTREQUEST	NEXTSTATION	NULLSTATION
PROCEDURE	PROCESS	PROTOCOL
PURGE	REBLOCK	RECEIVE
REPLY	RTERROR	SENDBLOCK
SHORTEN	SIGNAL	SKIP
STORE	STRING	STRUCTURE
TERMINALTYPE	THEN	TRANSFER

Reserved and Recognized Words

TRANSLATE	TRANSLATETABLE	TRANSMIT
UNLOCK	UNTIL	WAIT
WAITRESULTTYPE	WHILE	

Recognized Words in the Protocol Module

The following list contains the recognized words for the protocol module:

ABORTED	ABORTREASON	ACK
ADAPTOREVENT	ADAPTORREADY	ADDRESSERROR
ADDRESSMODE	AND	ANDM
ASCII	ASYNC	AUTOANSWER
BASIC	BCCEVEN	BCCODD
BCS	BCSERROR	BCSOF
BEL	BINARY	BITRATE
BREAK	BS	BSL
BUFFEROVERFLOW	BUSY	CAN
CANCELLED	CAND	CATEGORY
CB	CCITTEVEN	CCITTODD
CIMP	CLEARTOSEND	CODE
COMPLETED	CONNECTED	CONTROLMESSAGE
CONTROLMODE	COR	CR
CRC	CRC16	DATA CARRIER DETECT
DATASETREADY	DATATERMINALREADY	DC1
DC2	DC3	DC4
DEL	DESTINATION	DIAGNOSE
DIAGNOSTICRATE	DISCONNECT	DISCONNECTACTION
DIV	DLE	DLEDETECT
DROP	DYNAMIC	EBCDIC
ECHOPLEX	EM	ENABLED
ENDTEXT	ENQ	EOT
EQL	EQV	ERROR
ERRORDETECT	ESC	ETB
ETX	EVEN	EXTENDED
EXTERNAL	FALSE	FETCHPARITY
FF	FIRSTERROR	FLAGS
FORMATERROR	FRAME	FRAMEABORT

FRAMEDETECT	FS	FSL
FUNCTION	GEQ	GROUP
GS	GTR	HAPPENED
HEAD	HORIZONTAL	HT
IDLEDETECT	IMP	INHIBIT
INPROGRESS	INPUT	INPUTACTION
LENGTH	LEQ	LF
LINE	LINECHAR	LINECOUNT
LINEDIAL	LINEWIDTH	LONG
LOSSOFCARRIER	LSS	MANUALDIAL
MAX	MAXINPUT	MAXOUTPUT
MIN	MOD	NAK
NEQ	NOCANCEL	NOMEMORY
NONE	NOT	NOTIMEOUT
NOTM	NRZI	NULL
NULLREFERENCE	ODD	ONES
OR	ORM	OUTPUT
OVERRUN	PAGECOUNT	PAGESIZE
PARITY	POL	QUEUED
READY	RECEIVEADDRESS	RECEIVER
REJECTED	REPEAT	REQUEST
REQUESTLIST	REQUESTTOSEND	RESIDUE
RESIDUEERROR	RESULT	RESULTLIST
RESULTREQUIRED	RINGINDICATOR	ROTATELEFT
ROTATERIGHT	RS	SCREEN
SEL	SELECT	SEND
SENDANDWAIT	SHIFTLEFT	SHIFTRIGHT
SHORT	SI	SO
SOH	SOURCE	SP
SPECIAL	SPECIALINPUT	SPECIALOUTPUT
STATION	STATIONREFERENCE	STATUSCHANGED
STNSETDIAL	STNSETDIALANSWER	STOPBIT
STOPBITS	STRINGADD	STRINGSUBTRACT
STX	SUB	SUBSTRING
SUPRESSACTION	SYN	SYNC

Reserved and Recognized Words

SYNCCONTROL	SYSTEM	SYSTEMWAIT
TAIL	TAKE	TESTADAPTOR
TEXT	TEXTSIZE	TIMEINTERVAL
TIMEOUT	TIMESTAMP	TO
TRANSLATEIN	TRANSLATEOUT	TRANSMITADDRESS
TRANSMITTER	TRANSPARENT	TRIM
TRUE	TYPE	UNDERRUN
US	USAGECOUNT	VERTICAL
VT	WAITFORIDLE	WRAPAROUND
XOR	XORM	X1
X2	X3	X4
X5	X6	X7
X8	X9	X10
X11	X12	X13
X14	X15	X16

Reserved Words in the Configuration Module

The following list contains the reserved words for the configuration module:

ATTRIBUTE
DEFAULT
FILE
GROUP
HARDWARE
LINE
STATION
STATIONSET
TERMINAL

Appendix D

NDLII Language Components

This appendix presents a brief summary of the compiler options, declarations, and statements used in the NDLII language. These constructs are presented in the order in which they are normally used in the language. Whenever the order of appearance is not important, alphabetical order is used.

Recognized Compiler Control Options (\$ Options)

The following NDLII compiler control options are recognized:

CLEAR	LISTINCL	<sequence base option>
CODE	LISTP	<sequence increment option>
DELETE	MAP	SUMMARY
ERRORLIMIT	MERGE	TITLE
ERRORLIST	NEW	TRAINID
INCLUDE	PAGE	UPCASELISTING
LINEINFO	PAGESIZE	VERSION
LIST	SEQCHECK	VOID
LIST\$	SEQUENCE (SEQ)	WARNFATAL
LISTDELETED		XREF

General Information

The following declarations are allowed in a process:

BOOLEAN
BYTE
DEFINE
<enumerated variable>
ENUMERATION
INTEGER
PROCEDURE
STRING
STRUCTURE

The following declarations are allowed in a procedure:

BOOLEAN
BYTE
DEFINE
<enumerated variable>
ENUMERATION
INTEGER
STRING

The following elements are allowed in a structure:

BOOLEAN
BYTE
<enumerated variable>
INTEGER
STRING

The following are predeclared enumerated types:

CBCATEGORY
DISCONNECTPOLICY
INPUTPOLICY
RTERROR
WAITRESULTTYPE

The following are valid Boolean operators:

AND
CAND
CIMP
COR
EQV
IMP
NOT
OR
XOR

The following functions are general integer functions:

ANDM
LENGTH
MAX
MIN
NOTM
ORM
ROTATELEFT
ROTATERIGHT
SHIFTLEFT
SHIFTRIGHT
XORM

Protocol Module Declarations

The following declarations are allowed in the protocol module:

```
ABORTTYPE
ALGORITHM
APPLICATIONTYPE

CLASS      MODE = ASYNC  BITRATE
                                CODE
                                PARITY
                                STOPBITS

                                MODE = BIT  ADDRESSMODE
                                BITRATE
                                CODE
                                CONTROLMODE
                                NRZI

                                MODE = SYNC  CODE
                                DIAGNOSTICRATE
                                PARITY
                                SYNCCONTROL

DEFINE
EDITOR
ENUMERATION
INCLUDE (global)

REPLY      INPUT
                                (global)
                                OUTPUT

SIGNAL OUTPUT (global)
TERMINALTYPE
TRANSLATETABLE
```

Editor Declarations

The following declarations are allowed in editors:

```
BOOLEAN
BYTE
DEFINE
<enumerated variable>
ENUMERATION
INTEGER
PROCESS INPUT
PROCESS OUTPUT
STRING
STRUCTURE
```


Predeclared structures: REQUEST
 RESULT
 STATION

Predeclared variables: BUFFEROVERFLOW
 DESTINATION
 SOURCE

Editor Statements

The following statements are allowed in editors:

ABORT
ALLOCATE
<assignment statement>
CALL
CASE
<compound statement>
DO
IF
SHORTEN
SKIP
TRANSFER
TRANSLATE
WHILE

Special Functions: DIV
 DROP
 HEAD
 INTEGER
 MOD
 STRING
 STRINGADD
 STRINGSUBTRACT
 SUBSTRING
 TAIL
 TAKE
 TRANSLATEIN
 TRANSLATEOUT
 TRIM

Algorithm Declarations

The following declarations are allowed in algorithms:

ADAPTOR CONTROL
DEFINE
DUPLICATE STRUCTURE
ENUMERATION
LINE CONTROL
REPLY CB
SIGNAL CB

Line Control Declarations

The following declarations are allowed in line controls:

BOOLEAN
BYTE
CONTROL BLOCK
DEFINE
<enumerated variable>
ENUMERATION
EVENT
INCLUDE
INTEGER
INTERLOCK
PROCESS (PROCESS LINE is required)
STRING
STRUCTURE

Predeclared structures: <control block>
GROUP
LINE
STATION
STATION.REQUEST
STATION.RESULT

Predeclared variables: NOMEMORY
SYSTEMWAIT
TIMESTAMP

Line Control Statements

The following statements are allowed in line controls:

ABORT
ALLOCATE
<assignment statement>
CALL
CASE

CAUSE
CLEAR STATION
<compound statement>
COPYSTRING
COPYSTRUCTURE
DEALLOCATE
DEQUEUE
DISCARDRESULT
DO
ENQUEUE
FINISHREQUEST
IF
INITIATE
LOCK
MOVETEXT
NEWRESULT
NEXTREQUEST
NEXTSTATION
NULLSTATION
PROCESS
PURGE
REBLOCK
SENDBOST
SHORTEN
SKIP
UNLOCK
WAIT
WHILE

Special functions: DIV
DROP
HAPPENED
HEAD
INTEGER
MOD
NULLREFERENCE
STRING
STRINGADD
STRINGSUBTRACT
SUBSTRING
TAIL
TAKE
TIMEINTERVAL
TRANSLATEIN
TRANSLATEOUT
TRIM
WAIT

Adaptor Control Declarations

The following declarations are allowed in adaptor controls:

BOOLEAN
BYTE
DEFINE
<enumerated variable>
ENUMERATION
INTEGER
PROCESS INPUT
PROCESS OUTPUT
STRING
STRUCTURE

Predeclared structures: CB
RECEIVER
RECEIVER.ERROR
TRANSMITTER
TRANSMITTER.ERROR

Predeclared variables: BUFFEROVERFLOW (INPUT process only)
ENDTEXT (OUTPUT process only)
FETCHPARITY

Adaptor Control Statements

The following statements are allowed in adaptor controls:

ABORT
<assignment statement>
CALL
CANCEL
CASE
<compound statement>
DISABLE
DO
ENABLE
FETCH
IF
INITIALIZE
LOADSTRUCTURE
RECEIVE
SHORTEN
SKIP
STORE
TRANSMIT
WAIT (ADAPTOR)
WHILE

Special functions: BCSOF
TRANSLATEIN
TRANSLATEOUT
WAITFORIDLE

Configuration Module Definitions

The following definitions are allowed in the configuration module:

ATTRIBUTE
DEFAULT LINE
DEFAULT STATION
DEFAULT TERMINAL
FILE
GROUP
LINE
STATION
STATIONSET
TERMINAL
HARDWARE

ATTRIBUTE Definition

The following additional attributes can be defined in ATTRIBUTE definitions:

TERMINAL
STATION
LINE

File Definition

The following attributes are allowed in FILE definitions:

FILES
STATIONS
TITLE

GROUP Definition

The following attributes are allowed in GROUP definitions:

ALGORITHM
INITIALIZE
LINES

LINE Definition

The following attributes are allowed in LINE definitions:

DEFAULT
ALGORITHM
CLASS
DISCONNECTACTION
DPRDELAY
FUNCTION
INITIALIZE
LOSSOF CARRIER
PHONE
READY
RECEIVEDDELAY
SETDIGITDELAY
STATIONS
TIMEOUT
TRANSMITDELAY

STATION Definition

The following attributes are allowed in STATION definitions:

DEFAULT
ADDRESS
ALGORITHM
APPLICATIONLIST
CONTROL
EDITOR
ENABLED
INITIALIZE
INPUTACTION
MYUSE
OUTPUTLIMIT
READY
TERMINAL
TITLE

STATIONSET Definition

The following attributes are allowed in STATIONSET definitions:

PHONE
STATIONS

TERMINAL Definition

The following attributes are allowed in **TERMINAL** definitions:

DEFAULT
ADDRESSLENGTH
CLASS
LINEWIDTH
MAXINPUT
MAXOUTPUT
PAGECOUNT
PAGESIZE
RECEIVEDDELAY
SCREEN
TRANSMITDELAY
TYPE
WRAPAROUND

Appendix E

NDLIIANALYZER

This appendix describes the NDLIIANALYZER program, which is used to analyze NDLII structures.

NDLIIANALYZER is an ALGOL program that accesses a NIFII file and an NSP dump file to produce information concerning the status of an NDLII program. No attempt is made to interpret the values of the NDLII variables returned by NDLIIANALYZER.

NDLIIANALYZER can be initiated from CANDE by a statement of the following general form:

```
RUN *SYSTEM/NDLIIANALYZER("<NDLII analyzer options>");  
      VALUE = <dump number>; FILE NIF(TITLE = <NIFII title>)
```

The NDLII analyzer options are described under "NDLIIANALYZER Options" in this appendix.

File Names

The NDLIIANALYZER program uses two input files, the NIFII file and the NSP dump file, and one output file. The TITLE and INTNAME of the NIFII file is NIF. The TITLE and INTNAME of the output file is LINE. The NSP dump file has an INTNAME of DUMP and a default TITLE of DUMP/NSP/0000.

The title of the NSP dump file can be changed by either of two methods:

- Entering a dump number by means of the TASKVALUE task attribute
- File-equating the TITLE attribute of the file DUMP

If a dump number is specified in the TASKVALUE attribute, a file title is constructed by appending the prefix *DUMP/NSP/* to the specified dump number. For example, the following CANDE *RUN* command changes the TITLE of the file DUMP to DUMP/NSP/0110:

```
RUN *SYSTEM/NDLIIANALYZER("ALL");  
      VALUE = 110
```

The second method of altering the NSP dump file title is through file equation. For example, the following CANDE *RUN* command changes the TITLE attribute of the file DUMP to DUMP/NSP/0100:

```
RUN *SYSTEM/NDLIIANALYZER("ALL");  
      FILE DUMP(TITLE=DUMP/NSP/0100)
```


If the TITLE attribute of the file DUMP is file-equated, the TASKVALUE attribute is ignored. The first 4 consecutive digits in the last node of the resulting title, whether specified by the TASKVALUE attribute or by file equation, are used as the NSP ID.

The title of the NIFII file can be altered by file-equating the TITLE attribute of file NIF. For example, the following CANDE RUN command changes the title of the file NIF to SYS22073/NIF:

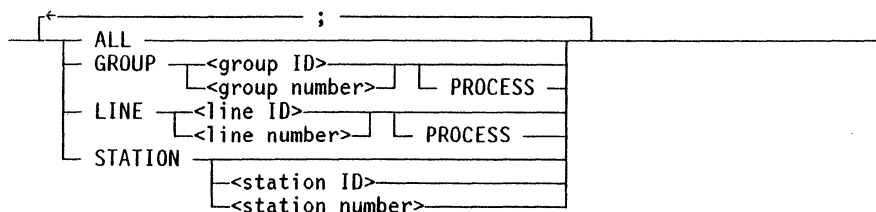
```
RUN *SYSTEM/NDLIIANALYZER("ALL");  
FILE NIF(TITLE=SYS22073/NIF)
```

NDLIIANALYZER Options

NDLIIANALYZER must be supplied with an option list that specifies the NDLII structures to be analyzed. The options are in the following hierarchy: ALL, GROUP, LINE, and STATION. Each higher-order option includes all of the lower-order options.

Syntax

<NDLII analyzer options>



Explanation

The ALL option produces an analysis of all groups, lines, and stations. The GROUP, LINE, and STATION options are subsets of this option.

The GROUP option produces an analysis of all lines and stations in the designated group. The group can be identified either by its group ID or by its group number.

The LINE option produces an analysis of all stations on the designated line. The line can be identified either by its line ID or by its line number.

The STATION option produces an analysis of the specified station. The station can be identified either by its station ID or by its station number. If no station is specified, all stations are analyzed.

The PROCESS option, when used with the GROUP or LINE option, produces an analysis of the processes associated with each line.

Example

The following CANDE *RUN* command initiates NDLIIANALYZER to analyze line TDLINE06, the associated line process, and station RJE2 for NSP 108. The file NIF is titled DATACOM/NIF, and the file DUMP is titled DUMP/NSP/0108.

```
RUN *SYSTEM/NDLIIANALYZER("STATION RJE2; LINE TDLINE06 PROCESS");
      VALUE=108; FILE NIF(TITLE=DATACOM/NIF)
```

Error Messages

Any of the errors listed among the error messages in Table E-1 cause NDLIIANALYZER to terminate after the corresponding message is displayed. Many of the errors listed in the table are caused by incompatibilities between the NSP dump file and the NIFII file. The errors are not necessarily related to an NDLI programming problem.

Table E-1. NDLIIANAYZER Error Messages

Message Number	Message
1	INVALID ABORTREASON IN CASE#1
2	INVALID CATEGORY IN CASE#2
3	NO DATATYPE SPECIFIED IN CASE#3
4	INVALID TAG IN CASE#4
5	INVALID TAG IN CASE#5
6	NO STRUCTURETYPE SPECIFIED IN CASE#6
7	NO STRUCTURETYPE SPECIFIED IN CASE#7
8	INVALID INPUTACTION TAG IN CASE#8
9	INVALID PROCESS STATE IN CASE#9
10	INVALID SWITCHED STATE IN CASE#10
11	INVALID DISCONNECTACTION IN CASE#11
12	INVALID CONTROL IN CASE#12
13	NO STRUCTURETYPE IN CASE#13
14	INVALID SHISTORY IN CASE#14
15	INVALID STYPE IN CASE#15
16	UNKNOWN DATATYPE <datatype>
17	NO MDT MATCH TO NIF NUMBER
18	DUMP NSP NOT IN NIF
19	VALID NSP NUMBER NOT FOUND IN TITLE
20	EDITOR NAME NOT FOUND

continued

Table E-1. NDLIIANAYZER Error Messages (cont.)

Message Number	Message
21	INVALID OPTION
22	UNKNOWN CHARACTER IN OPTIONS
23	NUMBER OR ALPHA STRING EXPECTED
24	NO OPTIONS FOUND
25	INVALID OPTION; SCANNING: <scanned character>
26	END OF COMMAND EXPECTED
27	END OF OPTIONS EXPECTED
28	>PROCESS< OR END OF OPTION EXPECTED
29	ID OR NUMBER EXPECTED
30	UNKNOWN NSP FIRMWARE
31	ENUMERATION NOT FOUND IN NIF LIST
32	INVALID SHISTORY
33	INVALID SABORTREASON

Errors 1 through 3, 8 through 11, 14, 15, 32, and 33 are the result of the inability of NDLIIANALYZER to equate the byte representing an enumeration value to a known enumeration.

Errors 4 and 5 occur when NDLIIANALYZER is building names for NDLI program-declared locks, events, and CBs included in a GROUP, LINE, or STATION structure. The error occurs if one of the previously included items is not an event, lock, or CB.

Error 6 is the result of the inability of NDLIIANALYZER to find an ID for a line or station in the NIFII file. Only groups and stationsets are allowed a null ID in the NIFII file.

Errors 7, 12, and 13 are internal software errors.

Error 16 occurs if an unknown data type is found in the NIFII file during the printing of included variables.

Error 17 is caused by the absence of a matching line, group, station, or stationset compiler-generated unique ID in the NIFII and NSP dump files.

Error 18 is caused by the absence of a matching NSP ID in the NIFII file for the NSP ID determined from the NSP dump file.

Error 19 occurs when the format of the NSP dump file title given in the file equation is not acceptable. Refer to “File Names” in this appendix for a description of the correct format.

Error 20 occurs when NDLIIANALYZER is unable to find a compiler-generated unique ID for an editor in the NIFII file to match that found by means of a process structure in the NSP dump file.

Errors 21 through 29 are the result of improper syntax in the NDLI analyzer options. Refer to “NDLIANALYZER Options” in this appendix for a description of these options.

Error 30 is caused by a version incompatibility between the NSP firmware and NDLIIANALYZER.

Error 31 occurs when NDLIIANALYZER cannot find a map value in the NIFII enumeration value list to correspond to the enumeration value in the NSP dump file.

Display Formats

The NDLIIANALYZER options designate a subset of the display formats to be used for output. Separate formats are available for groups, lines, processes, stationsets, and stations.

The output file, LINE, is a PRINTER file whose KIND must be file-equated to REMOTE as in the following example if screen output is desired:

```
FILE LINE(KIND=REMOTE)
```

For all string data (such as GROUPDATA or CB.SIGNAL), information is displayed only if the strings have been allocated. All other cases of possible omission are noted in the following formats, which list each format separately.

Group Display Format

GROUP <group ID>:

NUMBER: 4"0000"-4"FFFF", 0-65535
 LINECOUNT: 4"0000"-4"FFFF", 0-65535
 MAXINPUT: 4"0000"-4"FFFF", 0-65535

INCLUDED GROUP VARIABLES:

LOCAL NON EXTERNALS:

```
>> NO INCLUDED VARIABLES FOUND << /
<Boolean ID>: TRUE / FALSE
<byte ID>: 4"00"-4"FF", 0-255
<enumerated variable ID>(<enumerated constant>):
    4"00"-4"FF", 0-255
<event ID>: SET / RESET
<lock ID>: AVAILABLE / PROCURED
<integer ID>: 4"0000"-4"FFFF", 0-65535
<string ID>(MAX SIZE = 0-255, CURRENT SIZE = 0-255):
    <empty>/

    ( 0)  xx xx xx xx xx xx xx xx xx xx  "xxxxxxxxxx"
    ( 10) xx xx xx xx xx xx xx xx xx xx  "xxxxxxxxxx"
    . . .
```

GLOBAL NON EXTERNALS:

```
<all included variable information from previous NON EXTERNALS
format>
```

GLOBAL GROUP VARIABLES:

events, locks, and CBs from group structure extension

```
>> NO GLOBAL EXTENSION VARIABLES FOUND << /
EVENT @ (8, <offset>): SET / RESET

LOCK @ (8, <offset>): AVAILABLE / PROCURED
CB @ (8, <offset>):
    ABORTREASON: 4"00"-4"FF", 0-255
    CATEGORY: ABORTED / CANCELLED / COMPLETED / INPROGRESS /
              REJECTED
    TEXTSIZE: 4"0000"-4"FFFF", 0-65535
    TEXT:
        ( 0)  xx xx xx xx xx xx xx xx xx xx  "xxxxxxxxxx"
        ( 10) xx xx xx xx xx xx xx xx xx xx  "xxxxxxxxxx"
        . . .
    SIGNAL:
        ( 0)  xx xx xx xx xx xx xx xx xx xx  "xxxxxxxxxx"
        ( 10) xx xx xx xx xx xx xx xx xx xx  "xxxxxxxxxx"
```

```

. . .
REPLY:
( 0)  xx xx xx xx xx xx xx xx xx xx  "xxxxxxxxxx"
( 10) xx xx xx xx xx xx xx xx xx xx  "xxxxxxxxxx"
. . .
CB-ARRAY @ (8, <offset>):
[KEY]
<all of the CB information from the previous format>
[KEY]
<all of the CB information from the previous format>
. . .
GROUP.DATA
( 0)  xx xx xx xx xx xx xx xx xx xx  "xxxxxxxxxx"
( 10) xx xx xx xx xx xx xx xx xx xx  "xxxxxxxxxx"
. . .

```

Line Display Format

```

LINE <line ID>: LSP <unit ID>, mode <LSP mode>,
===== ADAPTOR <LSP adaptor number>, GROUP <group ID>

ALGORITHM: <algorithm ID>.<adaptor control ID>
NUMBER: 4"0000"-4"FFFF", 0-65535

<empty> / ATTACHED STATIONSET: <stationset ID>
ADAPTOREVENT: SET / RESET
ADAPTORREADY: TRUE / FALSE
SWITCHEDSTATE: UNINITIALIZED / PRIVATE / DISCONNECTED /
                RINGING / CONNECTED
CONNECTED: TRUE / FALSE
DISCONNECTACTION: AUTOANSWER / LINEDIAL / NONE / STNSETDIAL
                 STNSETDIALANSWER / MANUALDIAL
STATUSCHANGED: TRUE / FALSE
TOBEDELETED: TRUE / FALSE
BUSY: TRUE / FALSE
READY: TRUE / FALSE
SYSTEM: SET / RESET

INCLUDED GROUP VARIABLES:

LOCAL NON EXTERNALS:

>> NO INCLUDED VARIABLES FOUND << /
<Boolean ID>: TRUE / FALSE
<byte ID>: 4"00"-4"FF", 0-255
<enumerated variable ID>(<enumerated constant>):
    4"00"-4"FF", 0-255
<event ID>: SET / RESET
<lock ID>: AVAILABLE / PROCURED
<integer ID>: 4"0000"-4"FFFF", 0-65535
<string ID>(MAX SIZE = 0-255, CURRENT SIZE = 0-255):
    <empty>/

```

```
( 0)  xx xx xx xx xx xx xx xx xx "xxxxxxxxxx"
( 10)  xx xx xx xx xx xx xx xx xx "xxxxxxxxxx"
. . .
```

LOCAL EXTERNALS:

```
<all included variable information from previous NON EXTERNAL
format>
```

GLOBAL NON EXTERNALS:

```
<all included variable information from previous NON EXTERNAL
format>
```

GLOBAL EXTERNALS:

```
<all included variable information from previous NON EXTERNAL
format>
```

GLOBAL LINE VARIABLES:

events, locks, and CBs from line structure extension

>> NO GLOBAL EXTENSION VARIABLES FOUND << /

EVENT @ (8, <offset>): SET / RESET

LOCK @ (8, <offset>): AVAILABLE / PROCURED

CB @ (8, <offset>):

ABORTREASON: 4"00"-4"FF", 0-255

CATEGORY: ABORTED / CANCELLED / COMPLETED / INPROGRESS /
REJECTED

TEXTSIZE: 4"0000"-4"FFFF", 0-65535

TEXT:

```
( 0)  xx xx xx xx xx xx xx xx xx "xxxxxxxxxx"
( 10)  xx xx xx xx xx xx xx xx xx "xxxxxxxxxx"
. . .
```

SIGNAL:

```
( 0)  xx xx xx xx xx xx xx xx xx "xxxxxxxxxx"
( 10)  xx xx xx xx xx xx xx xx xx "xxxxxxxxxx"
. . .
```

REPLY:

```
( 0)  xx xx xx xx xx xx xx xx xx "xxxxxxxxxx"
( 10)  xx xx xx xx xx xx xx xx xx "xxxxxxxxxx"
. . .
```

CB-ARRAY @ (8, <offset>):

[KEY]

<all of the CB information from the previous CB format>

[KEY]

<all of the CB information from the previous CB format>

. . .

LINE.DATA

```
( 0)  xx xx xx xx xx xx xx xx xx xx  "xxxxxxxxxx"
( 10)  xx xx xx xx xx xx xx xx xx xx  "xxxxxxxxxx"
. . .
```

Process Display Format

PROCESS: LINE <line ID>, <process index>, MDTINX <process MDT inx>

DSED: TRUE / FALSE

STATE:

DELAY DELAYING FOR <integer> MILLISECONDS

EOJ END OF JOB PROCESSING

GETSPACE WAITING FOR SPACE

PROCURE WAITING FOR <lock ID>

READY READY

RUNNING RUNNING

STOPPED STOPPED

WAIT WAITING ON <event ID>

MULTIPLE-WAIT

<event ID>

<event ID>

. . .

TIMED-WAIT <integer> MILLISECONDS

<event ID>

<event ID>

. . .

TYPE: EDITOR (<editor ID>) /

LINE CONTROL (ALGORITHM: <algorithm ID>)

<empty> /

SHISTORY: ABORTED / FAULT TERMINATED / NORMAL TERMINATION /
EXECUTING

<empty> /

SABORTREASON: 4"00"-4"FF", 0-255 / PROGRAM STACK OVERFLOW /

PROGRAM STACK UNDERFLOW / STRING PROTECT /

STRUCTURE PROTECT / FETCH OVERFLOW /

RECEIVER NOT ENABLED / RECEIVER ALREADY ENABLED /

TRANSMITTER NOT ENABLED / INVALID S-OPERATOR /

TRANSMITTER ALREADY ENABLED / INVALID CASE VALUE /

TEXT AND STRUCTURE MISMATCH / INVALID ADDRESS /

INVALID SIGNAL/REPLY / INVALID OPERAND /

INVALID CALL / INVALID KERNEL CALL /

DIVIDE BY ZERO / EMPTY SOURCE / NULL STATION /

OCCUPIED DESTINATION / PROTECTED NAME

SUSPENDED: TRUE / FALSE

PROGRAM COUNTER:

<segment>:<offset> (<symbolic label>)

<empty> / SSTATION: <station ID>

<empty> / RETURN STACK:

<segment>:<offset> (<symbolic label>)

<segment>:<offset> (<symbolic label>)

. . .

PROCURED LOCKS:

>> EMPTY << /

<lock ID>

<lock ID>

. . .

<empty> / PARENT: MDTINX <process MDT index>

PROCESS.LOCALDATA:

(0) xx xx xx xx xx xx xx xx xx xx "xxxxxxxxxx"

(10) xx xx xx xx xx xx xx xx xx xx "xxxxxxxxxx"

. . .

STATIONSET Display Format

STATIONSET: <stationset ID>

NUMBER: 4"0000"-4"FFFF", 0-65535

Station Display Format

```

STATION <station ID>: LINE <line ID>
NUMBER: 4"0000"-4"FFFF", 0-65535
CONTROL: 4"00"-4"FF", 0-255
INPUTACTION: SEND / SENDANDWAIT
MAXINPUT: 4"0000"-4"FFFF", 0-65535
MAXOUTPUT: 4"0000"-4"FFFF", 0-65535
OUTPUTLIMIT: 4"0000"-4"FFFF", 0-65535
USAGECOUNT: 4"0000"-4"FFFF", 0-65535
READY: TRUE / FALSE
TOBECLEARED: TRUE / FALSE
TOBEDELETED: TRUE / FALSE
WAITINGACKNOWLEDGE: TRUE / FALSE
<empty> / STAEDITOR: <editor ID>
>> REQUEST NOT FOUND << /
REQUEST:
REQUESTNUMBER: 4"00000000"-4"FFFFFFFF", 0-(2**32-1)
TEXTSIZE: 4"0000"-4"FFFF", 0-65535
OUTPUTPRIORITY: 4"00"-4"FF", 0-255
SIGNAL:
( 0) xx xx xx xx xx xx xx xx xx "xxxxxxxxxx"
. . .
<Boolean ID>: TRUE / FALSE
<byte ID>: 4"00"-4"FF", 0-255
<enumerated variable ID>(<enumerated constant>):
    4"00"-4"FF", 0-255
<integer ID>: 4"0000"-4"FFFF", 0-65535
<string ID>(MAX SIZE = 0-255, CURRENT SIZE = 0-255):
    <empty>/
    ( 0) xx xx xx xx xx xx xx xx xx "xxxxxxxxxx"
    ( 10) xx xx xx xx xx xx xx xx xx "xxxxxxxxxx"
    . . .
REPLY:
( 0) xx xx xx xx xx xx xx xx xx "xxxxxxxxxx"
. . .
<Boolean ID>: TRUE / FALSE
<byte ID>: 4"00"-4"FF", 0-255
<empty>/
<enumerated variable ID>(<enumerated constant>):
    4"00"-4"FF", 0-255
<integer ID>: 4"0000"-4"FFFF", 0-65535
<string ID>(MAX SIZE = 0-255, CURRENT SIZE = 0-255)
    <empty>/
    ( 0) xx xx xx xx xx xx xx xx xx "xxxxxxxxxx"
    ( 10) xx xx xx xx xx xx xx xx xx "xxxxxxxxxx"
    . . .
<enumerated variable ID>(<enumerated constant>):
    4"00"-4"FF", 0-255
<integer ID>: 4"0000"-4"FFFF", 0-65535

<string ID>(MAX SIZE = 0-255, CURRENT SIZE = 0-255)

```

```

    <empty>/
    ( 0) xx xx xx xx xx xx xx xx xx xx "xxxxxxxxxx"
    ( 10) xx xx xx xx xx xx xx xx xx xx "xxxxxxxxxx"
    . . .
TEXT:
    ( 0) xx xx xx xx xx xx xx xx xx xx "xxxxxxxxxx"
    ( 10) xx xx xx xx xx xx xx xx xx xx "xxxxxxxxxx"
    . . .
REQUESTLIST:
    >> EMPTY << /
    REQUEST:TAG
    <all REQUEST information from the previous format>
    REQUEST:TAG
    <all REQUEST information from the previous format>
    . . .
<empty> /
RESULT:
    TEXTSIZE: 4"0000"-4"FFFF", 0-65535
    CONTROLMESSAGE: TRUE / FALSE
    SUPPRESSACTION: TRUE / FALSE
    TEXT:
    ( 0) xx xx xx xx xx xx xx xx xx xx "xxxxxxxxxx"
    ( 10) xx xx xx xx xx xx xx xx xx xx "xxxxxxxxxx"
    . . .
REPLY:
    ( 0) xx xx xx xx xx xx xx xx xx xx "xxxxxxxxxx"
    ( 10) xx xx xx xx xx xx xx xx xx xx "xxxxxxxxxx"
    . . .
    <Boolean ID>: TRUE / FALSE
    <byte ID>: 4"00"-4"FF", 0-255
    <empty>/
    <enumerated variable ID>(<enumerated constant>):
        4"00"-4"FF", 0-255
    <integer ID>: 4"0000"-4"FFFF", 0-65535
    <string ID>(MAX SIZE = 0-255, CURRENT SIZE = 0-255)
    <empty>/
    ( 0) xx xx xx xx xx xx xx xx xx xx "xxxxxxxxxx"
    ( 10) xx xx xx xx xx xx xx xx xx xx "xxxxxxxxxx"
    . . .

    <enumerated variable ID>(<enumerated constant>):
        4"00"-4"FF", 0-255
    <integer ID>: 4"0000"-4"FFFF", 0-65535
    <string ID>(MAX SIZE = 0-255, CURRENT SIZE = 0-255)
    <empty>/
    ( 0) xx xx xx xx xx xx xx xx xx xx "xxxxxxxxxx"
    ( 10) xx xx xx xx xx xx xx xx xx xx "xxxxxxxxxx"
    . . .
RESULTLIST:
    >> EMPTY << /
    RESULT: TAG
    <all RESULT information from the previous format>

```

```

RESULT: TAG
  <all RESULT information from the previous format>
  . . .

Q:
  >> EMPTY << /
  REQUEST: TAG
    <all REQUEST information from the previous format>
  REQUEST: TAG
    <all REQUEST information from the previous format>
    . . .

INCLUDED STATION VARIABLES:

  LOCAL NON EXTERNALS:

    >> NO INCLUDED VARIABLES FOUND << /
    <Boolean ID>: TRUE / FALSE
    <byte ID>: 4"00"-4"FF", 0-255
    <enumerated variable ID>(<enumerated constant>):
      4"00"-4"FF", 0-255
    <event ID>: SET / RESET
    <lock ID>: AVAILABLE / PROCURED
    <integer ID>: 4"0000"-4"FFFF", 0-65535
    <string ID>(MAX SIZE = 0-255, CURRENT SIZE = 0-255):
      <empty>/
      ( 0) xx xx xx xx xx xx xx xx xx xx "xxxxxxxxxx"
      ( 10) xx xx xx xx xx xx xx xx xx xx "xxxxxxxxxx"
      . . .

  LOCAL EXTERNALS:

    <all included variable information from previous
    LOCAL NON EXTERNAL format>

  GLOBAL NON EXTERNALS:

    <all included variable information from previous
    LOCAL NON EXTERNAL format>

  GLOBAL EXTERNALS:

    <all included variable information from previous
    LOCAL NON EXTERNAL format>

GLOBAL STATION VARIABLES:

events, locks, and CBs from station structure extension

  >> NO GLOBAL EXTENSION VARIABLES FOUND << /
  EVENT @ (8, <offset>): SET / RESET
  LOCK @ (8, <offset>): AVAILABLE / PROCURED

```

```

CB @ (8, <offset>):
  ABORTREASON: 4"00"-4"FF", 0-255
  CATEGORY: ABORTED / CANCELLED / COMPLETED / INPROGRESS /
            REJECTED
  TEXTSIZE: 4"0000"-4"FFFF", 0-65535
  TEXT:
    ( 0)  xx xx xx xx xx xx xx xx xx xx  "xxxxxxxxxx"
    ( 10) xx xx xx xx xx xx xx xx xx xx  "xxxxxxxxxx"
    . . .
  SIGNAL:
    ( 0)  xx xx xx xx xx xx xx xx xx xx  "xxxxxxxxxx"
    ( 10) xx xx xx xx xx xx xx xx xx xx  "xxxxxxxxxx"
    . . .

  REPLY:
    ( 0)  xx xx xx xx xx xx xx xx xx xx  "xxxxxxxxxx"
    ( 10) xx xx xx xx xx xx xx xx xx xx  "xxxxxxxxxx"
    . . .
CB-ARRAY @ (8, <offset>):
  [KEY]
    <all of the information from the previous CB format>
  [KEY]
    <all of the information from the previous CB format>
    . . .
STATION.DATA
  ( 0)  xx xx xx xx xx xx xx xx xx xx  "xxxxxxxxxx"
  ( 10) xx xx xx xx xx xx xx xx xx xx  "xxxxxxxxxx"
  . . .

```

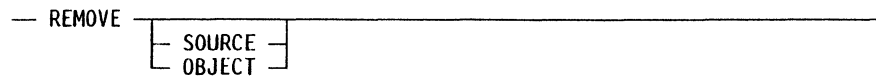
Appendix F

Understanding Railroad Diagrams

What Are Railroad Diagrams?

Railroad diagrams are diagrams that show you the rules for putting words and symbols together into commands and statements that the computer can understand. These diagrams consist of a series of paths that show the allowable structure, constants, and variables for a command or a statement. Paths show the order in which the command or statement is constructed. Paths are represented by horizontal and vertical lines. Many railroad diagrams have a number of different paths you can take to get to the end of the diagram.

Example



If you follow this railroad diagram from left to right, you will discover three acceptable commands. These commands are

REMOVE

REMOVE SOURCE

REMOVE OBJECT

If all railroad diagrams were this simple, this explanation could end here. However, because the allowed ways of communicating with the computer can be complex, railroad diagrams sometimes must also be complex.

Regardless of the level of complexity, all railroad diagrams are visual representations of commands and statements. Railroad diagrams are intended to

- Show the mandatory items.
- Show the user-selected items.
- Present the order in which the items must appear.
- Show the number of times an item can be repeated.
- Show the necessary punctuation.

To familiarize you with railroad diagrams, this explanation describes the elements of the diagrams and provides examples.

Some of the actual railroad diagrams you will encounter might be more complex. However, all railroad diagrams, simple or complex, follow the same basic rules. They

Understanding Railroad Diagrams

all consist of paths that represent the allowable structure, constants, and variables for commands and statements.

By following railroad diagrams, it is easy to understand the correct syntax for commands and statements. Once you become proficient in the use of railroad notation, the diagrams serve as quick references to the commands and statements.

Constants and Variables

A constant is an item that cannot be altered. You must enter the constant as it appears in the diagram, either in full or as an allowable abbreviation. If a constant is partially underlined, you can abbreviate the constant by entering only the underlined letters. In addition to the underlined letters, any of the remaining letters can be entered. If no part of the constant is underlined, the constant cannot be abbreviated. Constants can be recognized by the fact that they are never enclosed in angle brackets (< >) and are in uppercase letters.

A variable is an item that represents data. You can replace the variable with data that meets the requirements of the particular command or statement. When replacing a variable with data, you must follow the rules defined for the particular command or statement. Variables appear in railroad diagrams enclosed in angle brackets (< >).

In the following example, BEGIN and END are constants while <statement list> is a variable. The constant BEGIN can be abbreviated since it is partially underlined. Valid abbreviations for BEGIN are BE, BEG, and BEGI.

— BEGIN —<statement list>— END —————|

Constraints

Constraints are used in a railroad diagram to control progression through the diagram. Constraints consist of symbols and unique railroad diagram line paths. They include

- Vertical bars
- Percent signs
- Right arrows
- Required items
- User-selected items
- Loops
- Bridges

A description of each item follows.

Vertical Bar

The vertical bar symbol (|) represents the end of a railroad diagram and indicates the command or statement can be followed by another command or statement.

— SECONDWORD — (—<arithmetic expression>—) —————|

Percent Sign

The percent sign (%) represents the end of a railroad diagram and indicates the command or statement must be on a line by itself.

— STOP —————%

Right Arrow

The right arrow symbol (>) is used when the railroad diagram is too long to fit on one line and must continue on the next. A right arrow appears at the end of the first line and another right arrow appears at the beginning of the next line.

— SCALERIGHT — (—<arithmetic expression>— , —————→
→<arithmetic expression>—) —————|

Required Items

A required item can be either a constant, a variable, or punctuation. A required item appears as a single entry, by itself or with other items, on a horizontal line. Required items can also exist on horizontal lines within alternate paths or nested (lower-level) diagrams. If the path you are following contains a required item, you must enter the item in the command or statement; the required item cannot be omitted.

In the following example, the word EVENT is a required constant and <identifier> is a required variable:

— EVENT —<identifier>—————|

User-Selected Items

User-selected items appear one below the other in a vertical list. You can choose any one of the items from the list. If the list also contains an empty path (solid line), none of the choices are required. A user-selected item can be either a constant, a variable, or punctuation. In the following railroad diagram, either the plus sign (+) or minus sign (-) can be entered before the required variable <arithmetic expression>, or the symbols can be disregarded because the diagram also contains an empty path.

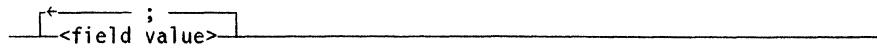
┌ ────┐ ────<arithmetic expression>—————|
└ + - ─┘

Loop

A loop represents an item or group of items that you can repeat. A loop can span all or part of a railroad diagram. It always consists of at least two horizontal lines, one below the other, connected on both sides by vertical lines. The top line is a right-to-left path that contains information about repeating the loop.

Understanding Railroad Diagrams

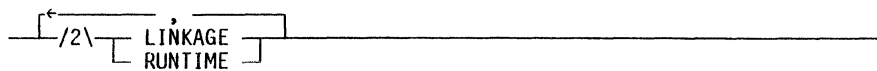
Some loops include a return character. A return character is a character (often a comma or semicolon) required before each repetition of a loop. If there is no return character, the items must be separated by one or more blank spaces.



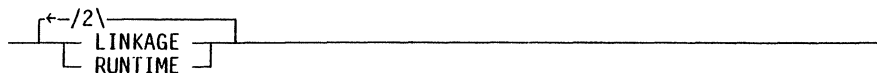
Bridge

Sometimes a loop also includes a bridge, which is used to show the maximum number of times the loop can be repeated. The bridge can precede the contents of the loop, or it can precede the return character (if any) on the upper line of the loop.

The bridge determines the number of times you can cross that point in the diagram. The bridge is an integer enclosed in sloping lines (/ \). Not all loops have bridges. Those that do not can be repeated any number of times until all valid entries have been used.

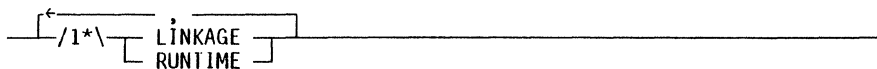


or



In the first bridge example, you can enter LINKAGE or RUNTIME no more than two times. In the second bridge example, you can enter LINKAGE or RUNTIME no more than three times.

In some bridges an asterisk follows the number. The asterisk means that you must select one item from the group.



The following figure shows the types of constraints used in railroad diagrams.

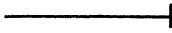
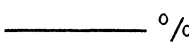
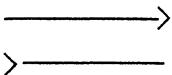
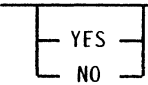
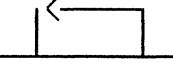
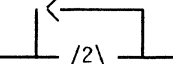
SYMBOL/PATH	EXPLANATION
	Vertical bar. Indicates that the command or statement can be followed by another command or statement.
	Percent sign. Indicates that the command or statement must be on a line by itself.
	Right arrow. Indicates that the diagram occupies more than one line.
	Required items. Indicates the constants, variables, and punctuation that must be entered in a command or statement.
	User-selected items. Indicates the items that appear one below the other in a vertical list. You select which item or items to include.
	A loop. Indicates an item or group of items that can be repeated.
	A bridge. Indicates the maximum number of times a loop can be repeated.

Figure F-1. Railroad Constraints

Following the Paths of a Railroad Diagram

The paths of a railroad diagram lead you through the command or statement from beginning to end. Some railroad diagrams have only one path, while others have several alternate paths. The following railroad diagram indicates there is only one path that requires the constant LINKAGE and the variable <linkage mnemonic>:

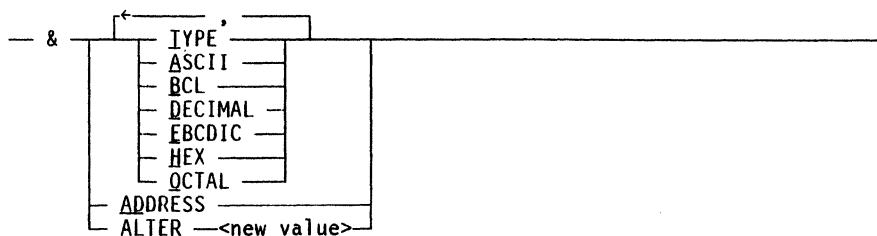
— LINKAGE —<linkage mnemonic>—|

Alternate paths provide choices in the construction of commands and statements. Alternate paths are provided by loops, user selected items, or a combination of both. More complex railroad diagrams can consist of many alternate paths, or nested (lower-level) diagrams, that show a further level of detail.

For example, the following railroad diagram consists of a top path and two alternate paths. The top path includes an ampersand (&) and the constants (that are

Understanding Railroad Diagrams

user-selected items) in the vertical list. These constants are within a loop that can be repeated any number of times until all options have been selected. The first alternate path requires the ampersand (&) and the required constant ADDRESS. The second alternate path requires the ampersand (&) followed by the required constant ALTER and the required variable <new value>.



Railroad Diagram Examples

The following examples show five railroad diagrams and possible command and statement constructions based on the paths of these diagrams.

Example 1

<lock statement>

— LOCK — (— <file identifier> —) —————|

Sample Input

LOCK (F1)

LOCK (FILE4)

Explanation

LOCK is a constant and cannot be altered. Because no part of the word is underlined, the entire word must be entered. The parentheses are required punctuation and F1 and FILE4 are sample <file identifier>s.

Example 2

<open statement>

— OPEN — [INQUIRY UPDATE] —<database name>—————|

Sample Input

OPEN DATABASE1

OPEN INQUIRY DATABASE1

OPEN UPDATE DATABASE1

Explanation

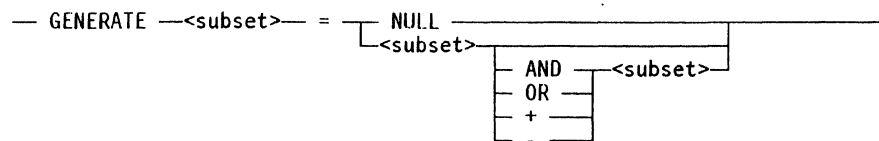
The first sample input shows the constant OPEN followed by the variable DATABASE1, which is a database name. The railroad diagram shows two user-selected items, INQUIRY and UPDATE. However, because there is an empty path (solid line), these entries are not required.

The second sample input shows the constant OPEN followed by the user-selected constant INQUIRY and the variable DATABASE1.

The third sample input shows the constant OPEN followed by the user-selected constant UPDATE and the variable DATABASE1.

Example 3

<generate statement>



Sample Input

GENERATE Z = NULL

GENERATE Z = X

GENERATE Z = X AND B

GENERATE Z = X + B

Explanation

The first sample input shows the GENERATE constant followed by the variable Z, an equal sign, and the user-selected constant NULL.

The second sample input shows the GENERATE constant followed by the variable Z, an equal sign, and the user-selected variable X.

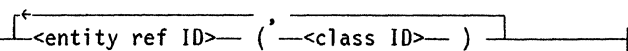
The third sample input shows the GENERATE constant followed by the variable Z, an equal sign, the user-selected variable X, the AND command (from the list of user-selected items in the nested path), and a third variable, B.

The fourth sample input shows the GENERATE constant followed by the variable Z, an equal sign, the user-selectable variable X, the plus sign (from the list of user-selected items in the nested path), and a third variable, B.

Understanding Railroad Diagrams

Example 4

<entity reference declaration>

— ENTITY REFERENCE — 

Sample Input

ENTITY REFERENCE ADVISOR1 (INSTRUCTOR)

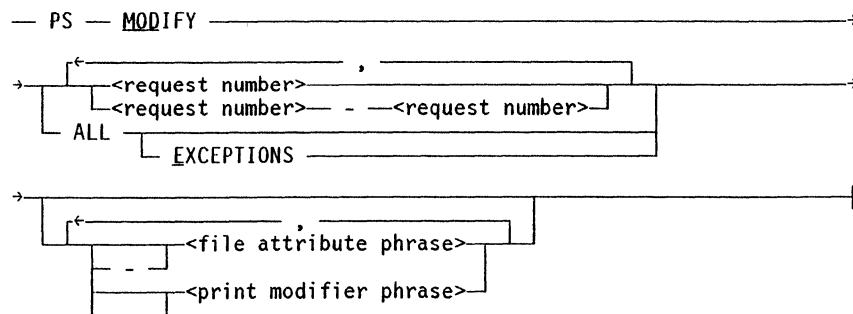
ENTITY REFERENCE ADVISOR1 (INSTRUCTOR), ADVISOR2 (ASST_INSTRUCTOR)

Explanation

The first sample input shows the required item ENTITY REFERENCE followed by the variable ADVISOR1 and the variable INSTRUCTOR. The parentheses are required.

The second sample input illustrates the use of a loop by showing the same input as in the first sample followed by a comma, the variable ADVISOR2, and the variable ASST_INSTRUCTOR. The parentheses are required.

Example 5

— PS — MODIFY — 

Sample Input

PS MODIFY 11159

PS MODIFY 11159,11160,11163

PS MODIFY 11159-11161 DESTINATION = "LP7"

PS MOD ALL EXCEPTIONS

Explanation

The first sample input shows the constants PS and MODIFY followed by the variable 11159, which is a <request number>.

The second sample input illustrates the use of a loop by showing the same input as in the first sample followed by a comma, the variable 11160, another comma, and the final variable 11163.

The third sample input shows the constants PS and MODIFY followed by the user-selected variables 11159-11161, which are <request number> s, and the user-selected variable DESTINATION = "LP7", which is a <file attribute phrase> .

The fourth sample input shows the constants PS and MODIFY followed by the user-selected constant ALL, followed by the user-selected constant EXCEPTIONS. Note that in this sample input, the constant MODIFY has been abbreviated.

Glossary

In this glossary definitions from the *Vocabulary for Data Processing, Telecommunications, and Office Systems* are preceded by VDP.

A

ACU

See automatic calling unit.

ACU interface

Signals and a signaling discipline used for communication between a line adapter and an automatic calling unit (ACU).

adapter control

In a Network Definition Language II (NDLII) program, the part of the protocol module that defines an input process and an output process to implement the real-time, low-level aspects of a line protocol through control of a line adapter.

address

(1) The identification of a location in storage (memory). (2) A sequence of bits, a character, or a group of characters that identifies a network station or a group of stations, a user, or an application. (3) The location of a device in the system configuration. (4) The identification of the location of a disk sector.

algorithm

(1) A sequence of instructions describing the steps needed to complete a particular task. (2) In Network Definition Language II (NDLII), the part of a program that contains the adapter control and line control processes for one type of line and specifies the line protocol for that type of line.

ASCII

American Standard Code for Information Interchange. A standard 7-bit or 8-bit information code used to represent alphanumeric characters, control characters, and graphic characters on a computer system.

async

See asynchronous transmission.

asynchronous transmission

A transmission mode in which the time of the occurrence of each character or block of characters is arbitrary. The characters, or blocks of characters, are surrounded by start and stop bits, from which a receiver derives the necessary timing for sampling bits. This mode is used for low- to medium-speed transmission of character strings. *Synonym for* start/stop transmission.

automatic calling unit (ACU)

A device that permits a modem or data terminal to place calls automatically and thereby establish a dialed link over the communication network. This device is used with data

circuit terminating equipment (DCE) to connect line adapters to data communications lines.

B

BCC

See block check character.

BCS

See block check sequence.

BDLC

See Data Link Control.

bisync procedure

A specific procedure that uses two synchronous bytes for synchronous transmission. The term *bisync* is a contraction of *binary synchronous*.

bit rate

The speed at which bits are transmitted over a communication channel.

bit-synchronous transmission mode

A transmission mode in which line synchronization is maintained by operating all data circuit terminating equipment (DCE) on the line at the same frequency and by keeping the DCEs in phase by framing each transmission with flag patterns. This mode is used for low- to high-speed transmission of arbitrary bit strings. On Unisys A Series systems, this mode is known as Data Link Control (BDLC); it is similar to synchronous data link control (SDLC).

block

(1) A group of physically adjacent records that can be transferred to or from a physical device as a group. (2) A program, or a part of a program, that is treated by the processor as a discrete unit. Examples are a procedure in ALGOL, a procedure or function in Pascal, a subroutine or function in FORTRAN, or a complete COBOL program.

block check

In data communications, an error-checking procedure applied to a block for control and recovery purposes. This procedure conforms to a predetermined set of rules agreed to at both ends of a communications channel.

block check character (BCC)

In data communications, a character appended to a data block that is used in block checking. Block checking is an error-checking procedure applied to a block for control and recovery purposes. This procedure conforms to a predetermined set of rules agreed to at both ends of a communications channel.

block check sequence (BCS)

The sequence of data resulting from the execution of a specific algorithm on transmitted data. This sequence is used to verify the validity of the transmitted data.

blocking factor

The number of logical records stored in a physical record on a disk or a tape.

broadband

A data communications channel capable of handling frequencies greater than those required for voice transmission (that is, frequencies from 50 KHz to 1.3 MHz). *Synonym for wideband.*

buffer

An area in which data is stored temporarily.

C**CANDE**

See Command and Edit language.

carrier

A continuous frequency that can be modulated, or impressed, with a second, information-carrying signal.

CB

See control block.

channel

(1) The physical or logical path allowing the transmission of information between a data source and a data sink or receiver. (2) In data communications, a communication path used to transmit signals between two or more points.

character

In data communications, 8 contiguous bits (1 byte).

circuit

A transmission medium connecting two or more electronic devices. A circuit is the configuration of equipment used in transmitting data from one location to another and can involve more than one type of facility.

code extension

A technique that allows the transportation of 8-bit character data through a 7-bit line protocol.

Command and Edit language (CANDE)

A time-sharing message control system (MCS) that enables a user to create and edit files, and develop, test, and execute programs, interactively.

configuration module

The major subdivision of a Network Definition Language II (NDLII) program that defines the structure of the data communications network and the physical and logical attributes of the lines and devices in the network.

connection

In data communications, the established path between two or more terminal installations. A connection can consist of one or more channels in tandem. A permanent connection is established without using switched facilities; a temporary connection is established using switched facilities.

contention mode

In data communications, a line condition in which no station is designated as a master station, and each station on the line must monitor signals on the line and wait for an idle condition before bidding for master status.

control block (CB)

On Entry and Medium Systems (EMS), a data structure used for communications between the host and the data communications subsystem, and between adapter control and line control.

control character

(1) Any extra transmitted character used to control or facilitate data transmission over communication networks or between data terminal equipment (DTE) units. (2) A character whose occurrence in a particular context initiates, modifies, or stops a control operation. Such a character can be recorded for use in a subsequent action; it can have a graphic representation.

D

data circuit terminating equipment (DCE)

In X.25, the functional unit of a data station that establishes, maintains, and releases a connection and provides the functions necessary for any code or signal conversion between the data terminal equipment (DTE) and the data transmission line. A DCE can be a separate piece of equipment. A DCE is the network supplier's equipment that can serve several user installations and is the user's entry point to the network.

Data Communications ALGOL (DCALGOL)

A Unisys language based on ALGOL that contains extensions for writing message control system (MCS) programs and other specialized system programs.

data communications controller (DCC)

The subset of the Master Control Program (MCP) operating as a group of independent tasks, each associated with one network support processor (NSP) or data communications data link processor (DCDLP).

data communications data link processor (DCDLP)

A data communications processor (DCP) that combines the functions of a network support processor (NSP) and a line support processor (LSP) into one physical data link processor (DLP) and supports up to four lines of communication.

data communications host adapter (DCHA)

A data communications processor on Micro A systems that combines the functions of a network support processor (NSP) and a line support processor (LSP) into one physical board and supports up to four lines of communication.

Data Link Control (BDLC)

The bit-synchronous transmission mode used on Unisys A Series systems. BDLC is compatible with the international standard High-level Data Link Control (HDLC) protocol.

data link processor (DLP)

A processor that serves as the system interface to a specific peripheral device, controller, or communications network.

data terminal equipment (DTE)

The functional unit of a data station that establishes, maintains, and releases a connection and provides code and signal conversion between the data station and the transmission line. A DTE can serve as a data source, a data sink, or both and can provide for the data communications control functions to be performed in accordance with link protocol. Packet-mode DTEs divide the data into packets. Non-packet-mode DTEs require packet assemblers/disassemblers (PADs) to operate in an X.25 message control system (MCS) environment.

DATACOMINFO file

A file that contains a complete description of the data communications configuration, including algorithms, editors, and translation tables. This is the file that the Interactive Datacomm Configurator (IDC) modifies and from which the Master Control Program (MCP) initializes the data communications subsystem.

DCALGOL

See Data Communications ALGOL.

DCC

See data communications controller.

DCDLP

See data communications data link processor.

DCE

See data circuit terminating equipment.

DCHA

See data communications host adapter.

DCWRITE

A system intrinsic that passes a specified message to the data communications controller (DCC).

dial-up

The process of, or the equipment or facilities involved in, using a dial or push-button data set (such as a telephone) to establish a temporary connection through a switched telephone network. Dial-up is also referred to as dial-in.

digit present (DPR)

In data communications, a signal from a line support processor (LSP) adapter to an automatic calling unit (ACU) indicating that the next number in the dialing sequence has been loaded.

Glossary

DLP

See data link processor.

DPR

See digit present.

drop

In data communications, a connection made available for a terminal on a line.

DTE

See data terminal equipment.

E

EBCDIC

Extended Binary Coded Decimal Interchange Code. An 8-bit code representing 256 graphic and control characters that are the native character set of most mainframe systems.

EDCDLP

See enhanced data communications data link processor.

editor

A Network Definition Language II (NDLII) program module that defines an input process and an output process. These processes implement application-dependent editing on the text portions of input and output messages according to the requirements of specific terminal types.

EMS

See Entry and Medium systems.

end of job (EOJ)

The termination of processing of a job.

end of task (EOT)

The termination of processing of a task.

enhanced data communications data link processor (EDCDLP)

A data communications processor (DCP) that combines the functions of a network support processor (NSP) and a line support processor (LSP) into one physical data link processor (DLP) and supports up to eight lines of communication. The EDCDLP is an upgrade from the data communications data link processor (DCDLP); it has a faster processor and a larger amount of usable random-access memory (RAM).

Entry and Medium Systems (EMS)

A designation referring to the Micro A and A 1 through A 10 systems.

EOJ

See end of job.

EOT

See end of transmission.

executive

In the Network Definition Language II (NDLII), the set of processes and procedures within the network support processor (NSP) operating system that manages request and result messages between the host and the NSP, and I/O operations between the NSP and its line support processors (LSPs).

F**FCS**

See frame check sequence.

FDX

See full duplex.

frame check sequence (FCS)

In bit-synchronous mode, the pattern used to verify the validity of transmissions delimited by a flag pattern.

full duplex (FDX)

In data communications, a type of transmission in which information can be sent in both directions at the same time. *Synonym for two-way simultaneous.*

H**half duplex (HDX)**

In data communications, a type of transmission in which information can be sent in one direction or the other, but not in both directions simultaneously. *Synonym for two-way alternate.*

HDX

See half duplex.

header

(1) A data structure that contains information about a disk file, such as the physical location of the file on the disk and various file attributes. A header is also referred to as a disk file header. (2) A sequence of characters preceding the text of a message, containing routing or other communications-related information.

I**I/O**

Input/output. An operation in which the system reads data from or writes data to a file on a peripheral device such as a disk drive.

I/O processor (IOP)

A specialized processor for moving data between system memory and the I/O subsystem.

Glossary

IDC

See Interactive Datacomm Configurator.

Interactive Datacomm Configurator (IDC)

A Unisys interactive, menu-driven utility that enables the user to create, interrogate, and modify data communications network configurations.

IOP

See I/O processor.

L

leased line

A connection established without the use of switching facilities for the exclusive use of two or more data stations. A leased line is also referred to as a leased channel, leased circuit, leased connection, dedicated circuit, dedicated channel, or dedicated line. *See* private line.

line

(1) A data transmission link between two computers or between a computer and its associated terminals. (2) In Network Definition Language II (NDLII), a logical construct representing a particular line adapter and all data structures associated with that line adapter. (3) In the Interactive Datacomm Configurator (IDC), a data structure that describes a physical line in the configuration.

line adapter

The device that performs physical line control, including byte assembly and disassembly, data buffering, data synchronization, and modem and autocal equipment control.

line control

The part of a Network Definition Language II (NDLII) program that defines one or more processes that control the execution of a particular line protocol by initiating adapter control processes to implement a particular station-management policy.

line support processor (LSP)

The data communications subsystem processor that manages communication with the host and initiates processes that control the input of messages to and the output of messages from data communications lines.

logical station number (LSN)

In Network Definition Language II (NDLII), a unique number assigned to each station in a network. Each station has an LSN assigned according to the order in which the stations are defined in NDLII. The first defined station is 0000.

LSN

See logical station number.

LSP

See line support processor.

M

Master Control Program (MCP)

An operating system on A Series systems. The MCP controls the operational environment of the system by performing job selection, memory management, peripheral management, virtual memory management, dynamic subroutine linkage, and logging of errors and system utilization.

MCP

See Master Control Program.

MCS

See message control system.

message

In data communications, any information-containing data unit, in an ordered format, sent by means of a communications process to a named network entity or interface. A message contains the information (text portion) and controls for routing and handling (header portion).

message control system (MCS)

A program that controls the flow of messages between terminals, application programs, and the operating system. MCS functions can include message routing, access control, audit and recovery, system management, and message formatting.

message level interface (MLI)

The interface between the host system, the I/O subsystem, and the data communications subsystem.

message level interface processor (MLIP)

See I/O processor, Entry and Medium Systems.

MLI

See message level interface.

MLIP

An acronym for the obsolete term message level interface processor; *see* I/O processor, Entry and Medium Systems.

modem

A device placed between a computer system and a telephone line to permit transmission of digital pulses. Modems permit computers to communicate with other computers, terminals, and printers over communication lines. The term *modem* is derived from *modulator-demodulator*.

multidrop

A data communications subsystem capable of controlling two or more connection endpoints on the same physical line.

N

NDLII

See Network Definition Language II.

NDLII post compiler (NPC)

A program that allows users to use the Network Definition Language II (NDLII) to develop protocols for the enhanced data communications data link processor (EDCDLP) and the data communications host adapter (DCHA).

Network Definition Language II (NDLII)

The Unisys language used to describe the physical, logical, and functional characteristics of the data communications subsystem to network support processors (NSPs), line support processors (LSPs), and data communications data link processors (DCDLPs).

network information file II (NIFI)

The file generated when a Network Definition Language II (NDLII) program is compiled. This file contains line support processor (LSP) and network support processor (NSP) code, data structures, and other information. A NIFI is also generally referred to as a network information file (NIF).

network support processor (NSP)

A data communications subsystem processor that controls the interface between a host system and the data communications peripherals. The NSP executes the code generated by the Network Definition Language II (NDLII) compiler for line control and editor procedures. An NSP can also control line support processors (LSPs).

NIFI

See network information file II.

node

(1) A data structure that consists of a list, a set of properties, and a block part. Either the list or the properties can be absent. (2) In a data communications network, a point at which one or more functional units interconnect with data transmission lines.

nonreturn to zero inverted (NRZI)

The magnetic recording of binary digits such that ones are represented by a change in the condition of magnetization, and zeros are represented by the absence of change. This method is called mark recording because only the one or mark signals are explicitly recorded.

NPC

See NDLII post compiler.

NRZI

See nonreturn to zero inverted.

NSP

See network support processor.

O

ODT

See operator display terminal.

operator display terminal (ODT)

A terminal or other device that is connected to the system in such a way that it can communicate directly with the operating system. The ODT allows operations personnel to accomplish system operations functions through either of two operating modes: system command mode or data comm mode.

P

parity bit

A bit appended to an array of bits to force the sum of all bits in the array to be odd (for odd parity) or even (for even parity), as required by the convention established.

parity check

An error-detection method that tests whether the number of ones or zeros in an array of binary digits is odd or even. Vertical parity checks verify the validity of individual characters in a block. Horizontal parity checks use the appended block check character (BCC) to verify the cumulative validity of all preceding characters in a block.

path

In Network Definition Language II (NDLII) and X.25, a route between two nodes.

PND

See present next digit.

point-to-point connection (PTP)

In data communications, a connection established between two data stations.

polling

A communications control procedure in which a master station systematically invites tributary stations on a multipoint circuit to transmit data.

present next digit (PND)

In data communications, the signal from the automatic calling unit (ACU) to a line support processor (LSP) adapter indicating that the ACU is ready to receive the next digit in the dialing sequence.

protocol module

In Network Definition Language II (NDLII), the major subdivision of a program that defines editor processes, line control processes, and adapter control processes.

PTP

See point-to-point connection.

R

reply area

In Network Definition Language II (NDLII), the portion of a control block used by adapter control to pass information to line control regarding the result of an operation, and used by line control to pass this information to the host.

S

SDLC

See synchronous data link control.

selection

In data communications, a short message transmitted to a terminal to determine whether the terminal is ready to accept data.

serial transmission

In data communications, sequential transmission of individual bits constituting a character or other entity of data.

signal area

In Network Definition Language II (NDLII), the portion of a control block used by the host to pass information to line control specifying an operation to be performed, and used by line control to pass this information to adapter control.

start bit

In data communications, a signal transmitted for the duration of one bit-time that indicates the beginning of a character.

station

In data communications, a data structure that relates a logical abstraction to either a physical device or a pseudostation.

stationset

In data communications, a grouping of stations by a remote site that enables either one switched line or a group of switched lines to easily control stations at a number of remote sites.

status

In data communications, the condition reflecting the ability of a station, at a given instant, to transmit or receive data.

stop bit

In data communications, a signal transmitted for the duration of one or two bit-times that indicates the end of a character.

switched line

In data communications, a communications link for which the physical path, established by dialing, can vary with each use.

synchronous data link control (SDLC)

(VDP) A discipline for managing synchronous, code-transparent, serial-by-bit information transfer over a line connection. Transmission exchanges can be duplex or half-duplex over switched or nonswitched links. The configuration of the link connection can be point-to-point, multipoint, or loop. SDLC conforms to subsets of the Advanced Data Communication Control Procedures of the American National Standards Institute and high-level data link control (HDLC) of the International Standards Organization (ISO). *See also* bit-synchronous transmission mode.

synchronous transmission

In data communications, a mode of data transmission in which each bit of data is transmitted at a frequency determined by an external clock source.

T**terminal**

An I/O device designed to receive or send source data in a network.

text

In data communications, the part of a message containing information that has an ultimate purpose and destination beyond the data communications subsystem.

throughput

The total useful information processed during a specified time period.

turnaround

In data communications, the operation of reversing the direction of transmission from send to receive or receive to send. Turnaround time is the time required to perform this operation.

V**voice-transmission frequencies**

In data communications, frequencies in the range of 300 to 3000 hertz.

W**WFL**

See Work Flow Language.

wideband

A data communications channel whose bandwidth is wider than a voice-grade channel. *Synonym for* broadband.

Work Flow Language (WFL)

A Unisys language used for constructing jobs that compile or run programs on A Series systems. WFL includes variables, expressions, and flow-of-control statements that offer the programmer a wide range of capabilities with regard to task control.

Bibliography

A Series Data Communications Protocols Installation and Implementation Guide
(form 8600 0486). Unisys Corporation.

A Series DCALGOL Programming Reference Manual (form 8600 0841). Unisys Corporation.

A Series Interactive Datacomm Configurator (IDC) Operations Guide (form 1169810). Unisys Corporation.

A Series System Commands Operations Reference Manual (form 8600 0395). Unisys Corporation.

A Series Work Flow Language (WFL) Programming Reference Manual
(form 8600 1047). Unisys Corporation.

Vocabulary for Data Processing, Telecommunications, and Office Systems
(form GC10-1699-6). Poughkeepsie, New York: International Business Machines Corporation, 1981.

Index

A

- <abort reason>, 2-35
- ABORT statement, 2-35
- <abort statement>, 2-35
- ABORTREASON control block variable, 3-39
- ABORTTYPE declaration, 3-3
- <adaptor body>, 3-79
- adaptor control
 - declarations, 3-80
 - definition, 3-78
 - fault termination, 3-107
 - functions, 3-104
 - predeclared variables, 3-88
 - BUFFEROVERFLOW, 3-88
 - ENDTEXT, 3-88
 - FETCHPARITY, 3-88
 - NOCANCEL, 3-88
 - process declarations, 3-88
 - processes, 3-78
 - responsibilities, 1-8
 - special statements, 3-89
 - WAIT statement, 3-103
- <adaptor control ID>, 3-79
- <adaptor control operation>, 3-61
- <adaptor control structure name>, 3-80
- ADAPTOR CONTROL, definition, 3-79
- <adaptor control>, 3-79
- <adaptor declaration list>, 3-79
- <adaptor translate function>, 3-105
- <adaptor wait statement>, 3-103
- ADAPTOREVENT LINE structure variable, 3-42
- ADAPTORREADY LINE structure variable, 3-42
- <add result length>, 3-30
- ADDRESS attribute, 4-21
- <address mode>, 3-9
- ADDRESSLENGTH attribute, 4-33
- ADDRESSMODE line class attribute, 3-9
- algorithm
 - components, 1-8
 - declaration, 3-34
 - definition, 1-7
 - description, 3-34
- ALGORITHM attribute
 - for groups, 4-7
 - for lines, 4-10
 - for stations, 4-22
- <algorithm declaration list>, 3-34
- <algorithm declaration>, 3-34
- <algorithm ID>, 3-34
- ALLOCATE
 - editor statement, 3-24
 - statement, 3-56
- <allocate statement>, 3-56
- <application editor list>, 4-23
- <application key>, 4-23
- APPLICATIONLIST attribute, 4-23
- APPLICATIONTYPE declaration, 3-3
- <arithmetic operator>, 2-27
- assignment statement, 2-36
 - Boolean, 2-36
 - byte, 2-36
 - enumerated, 2-37
 - integer, 2-38
 - string, 2-38
- <assignment statement>, 2-36
- <async class>, 3-5
- asynchronous mode, 3-5
- <attribute body>, 4-2
- ATTRIBUTE definition in configuration
 - module, 4-2
 - summary, D-8
- <attribute definition>, 4-2

B

- BCS RECEIVER structure variable, 3-81
- BCS TRANSMITTER structure variable, 3-85

- <BCS type>, 3-13
- BCSOF function, 3-104
- <BCSOF function>, 3-104
- <binary digit>, definition, 2-2
- <bit class>, 3-6
- bit mode, 3-6
- <bit rate value>, 3-10
- bit rate values, 3-10
- <bit rate>, 3-10
- BITRATE line class attribute, 3-10
- Boolean assignment, 2-36
- <Boolean assignment>, 2-36
- <Boolean attribute>, 4-2
- <Boolean constant>, definition, 2-4
- BOOLEAN declaration, 2-6
- <Boolean declaration>, 2-6
- Boolean expression, 2-21
- <Boolean expression>, 2-21
- <Boolean ID>, 2-6
- <Boolean initial value>, 2-6
- <Boolean option>, B-3
- <Boolean primary>, 2-21
- <Boolean target>, 2-36
- <Boolean variable>, 2-6
- <break time>, 3-99
- <buffer size>, 3-56
- BUFFEROVERFLOW
 - adaptor control variable, 3-88
 - editor variable, 3-24
- BUSY LINE structure variable, 3-42
- byte
 - assignment, 2-36
 - expression, 2-25
- <byte assignment>, 2-37
- <byte attribute>, 4-2
- <byte bit number>, 2-36
- <byte constant>, definition, 2-4
- BYTE declaration, 2-7
- <byte declaration>, 2-7
- <byte expression>, 2-25
- <byte ID>, 2-7
- <byte initial value>, 2-7
- <byte mnemonic>, definition, 2-4
- <byte primary>, 2-25
- <byte target in string>, 2-37
- <byte target>, 2-37
- <byte variable>, 2-7
- byte-to-integer promotion, 2-28
- <byte-to-integer promotion>, 2-28

C

- CALL statement, 2-38
- <call statement>, 2-38
- CANCEL statement, 3-89
- <cancel statement>, 3-89
- CANDE, (See Command and Edit (CANDE))
- <case body>, 2-39
- <case guard>, 2-39
- <case selector>, 2-39
- CASE statement, 2-39
- <case statement>, 2-39
- CATEGORY control block variable, 3-39
- CAUSE statement, 3-56
- <cause statement>, 3-56
- CB, (See control block (CB))
- <CB array ID>, 3-51
- <CB declaration>, 3-51
- <CB ID>, 3-51
- <CB limit>, 3-51
- <CB subscript>, 3-52
- <CB variable>, 3-51
- <CB-to-CB>, 3-66
- <CB-to-request>, 3-66
- <CB-to-result>, 3-66
- CBCATEGORY enumerated type, 2-11
- character set
 - definition, 2-34
 - mapping, 3-19
- <character set>, 2-34
- <character size>, 3-19
- <character>
 - definition, 2-2
 - in adaptor control RECEIVE statement, 3-95
- CLASS
 - attribute
 - for lines, 4-10
 - for terminals, 4-33
 - declaration, 3-4
- <class declaration>, 3-4
- <class ID>, 3-4
- CLEAR option in compiler control, B-4
- <clear option>, B-5
- CLEAR STATION statement, 3-57
- <clear station statement>, 3-57
- <cm definition list>, 4-1
- CODE line class attribute, 3-10
- CODE option in compiler control, B-5
- <code option>, B-5
- <code>, 3-10

- CODE_EXT_TYPE enumerated type, 2-11
- Command and Edit (CANDE), B-19
- compiler, 4-20, B-1
 - control options, B-4
 - CLEAR, B-4
 - CODE, B-5
 - DELETE, B-5
 - ERRORLIMIT, B-5
 - ERRORLIST, B-6
 - INCLUDE, B-6
 - LINEINFO, B-7
 - LIST\$, B-8
 - LISTDELETED, B-8
 - LISTINCL, B-8
 - LISTP, B-8
 - MAP, B-9
 - MERGE, B-9
 - NEW, B-10
 - PAGE, B-10
 - PAGESIZE, B-10
 - SEQUENCE, B-11
 - SUMMARY, B-12
 - summary of, D-1
 - TITLE, B-13
 - TRAINID, B-13
 - UPCASELISTING, B-14
 - VERSION, B-14
 - VOID, B-15
 - WARNFATAL, B-15
 - XREF, B-16
 - control records, B-1, B-2
 - error reporting, 1-6
 - files, B-1
 - input record format, 1-5
 - <compiler control record>, B-3
- complement operator, 2-23
 - precedence, 2-24
- <complement operator>, 2-22
 - precedence, 2-24
 - truth table, 2-23
- <complex string expression>, 2-33
- <complex string primary>, 2-33
- compound statement, 2-40
 - <compound statement>, 2-40
- concatenation, 2-32
 - <concatenation operator>, 2-33
- conditional operator, 2-23
 - precedence, 2-24
- <conditional operator>, 2-22
 - precedence, 2-24
 - truth table, 2-23
- configuration module, 4-1
- ATTRIBUTE definition, 4-2
- default definitions, 4-4
 - DEFAULT LINE, 4-4
 - DEFAULT STATION, 4-4
 - DEFAULT TERMINAL, 4-5
- definition, D-8
- description, 4-1
- example, 4-42
- FILE definition, 4-6
- group attributes
 - ALGORITHM, 4-7
 - INITIALIZE, 4-8
 - LINE ID, 4-8
- GROUP definition, 4-7
- HARDWARE DEFINITION, 4-39
- line attributes
 - ALGORITHM, 4-10
 - CLASS, 4-10
 - DISCONNECTACTION, 4-11
 - DPRDELAY, 4-11
 - FUNCTION, 4-12
 - INITIALIZE, 4-13
 - LOSSOF CARRIER, 4-14
 - PHONE, 4-15
 - READY, 4-16
 - RECEIVEDDELAY, 4-16
 - SETDIGITDELAY, 4-17
 - STATION list, 4-18
 - TIMEOUT, 4-19
 - TRANSMITDELAY, 4-19
- line DEFAULT specification, 4-9
- LINE definition, 4-9
- reserved words, list of, C-4
- station attributes
 - ADDRESS, 4-21
 - ALGORITHM, 4-22
 - APPLICATIONLIST, 4-23
 - CONTROL, 4-24
 - EDITOR, 4-24
 - ENABLED, 4-25
 - INITIALIZE, 4-25
 - INPUTACTION, 4-26
 - MYUSE, 4-26
 - OUTPUTLIMIT, 4-27
 - READY, 4-27
 - TERMINAL, 4-28
 - TITLE, 4-28
- station DEFAULT specification, 4-21
- STATION definition, 4-20
- stationset attributes
 - member list, 4-30
 - PHONE, 4-31

STATIONSET definition, 4-29
structure, 1-4
terminal attributes
 ADDRESSLENGTH, 4-33
 CLASS, 4-33
 LINEWIDTH, 4-34
 MAXINPUT, 4-34
 MAXOUTPUT, 4-35
 PAGECOUNT, 4-35
 PAGESIZE, 4-36
 RECEIVEDELAY, 4-36
 SCREEN, 4-37
 TRANSMITDELAY, 4-37
 TYPE, 4-38
 WRAPAROUND, 4-38
terminal DEFAULT specification, 4-32
TERMINAL definition, 4-32
<configuration module>, 4-1
configuration overview
 post compiler, B-18
CONNECTED LINE structure variable, 3-42
constant
 <Boolean constant>, 2-4
 <byte constant>, 2-4
 <byte mnemonic>, 2-4
definition, 2-4
 <EBCDIC string>, 2-5
 <hex string>, 2-5
 <integer constant>, 2-5
 <null string>, 2-5
 <numeric constant>, 2-5
 <string constant>, 2-5
 <constant Boolean expression>, 2-22
 <constant Boolean primary>, 2-22
 <constant byte expression>, 2-25
 <constant integer expression>, 2-27
 <constant integer primary>, 2-27
 <constant>, definition, 2-2
CONTROL attribute, 4-24
control block (CB), 1-9
 predeclared structure variables, 3-39
 ABORTREASON, 3-39
 CATEGORY, 3-39
 DCE, 3-39
 TEXT, 3-40
 TEXTSIZE, 3-40
control block (CB) structure
 predeclared variables, 3-81
CONTROL BLOCK declaration, 3-51
 <control mode>, 3-11
CONTROL STATION structure variable,
 3-45

CONTROLMODE line class attribute, 3-11
COPYSTRING statement, 3-57
 <copystring statement>, 3-57
COPYSTRUCTURE statement, 3-58
 <copystructure statement>, 3-58
 <cycle number>, B-14

D

data comm subsystem
 modules, 1-7
 overview, 1-7
 software requirements, 1-9
data communications controller (DCC), 1-9
data type
 classes, 2-1
 definition of, 2-1
DCC, (See data communications controller
 (DCC))
DCE control block variable, 3-39
DEALLOCATE statement, 3-58
 <deallocate statement>, 3-58
declarations
 ABORTTYPE, 3-3
 adaptor control, 3-80
 list, D-7
 algorithm, 3-34
 list, D-5
 APPLICATIONTYPE, 3-3
 BOOLEAN, 2-6
 BYTE, 2-7
 CLASS, 3-4
 CONTROL BLOCK, 3-51
 DEFINE, 2-8
 definition of, 2-1
 DUPLICATE structure, 3-35
 editor, 3-23
 list, D-3
 enumerated variable, 2-13
 ENUMERATION, 2-10
 EVENT, 3-52
 general, 2-6
 INCLUDE, 3-15
 INTEGER, 2-14
 INTERLOCK, 3-53
 line control, 3-38
 list, D-5
 PROCEDURE, 2-15
 list, D-2
 PROCESS, 2-16
 in line control, 3-54

- summary, D-1
- protocol module, list of, D-3
- REPLY, 3-16, 3-35
- SIGNAL, 3-16, 3-35
- special, 2-6
- STRING, 2-17
- STRUCTURE, 2-18
 - in line control, 3-54
- TERMINALTYPE, 3-18
- TRANSLATETABLE, 3-19
- variable, 2-19
- default definitions in configuration module, 4-4
- DEFAULT LINE definition, 4-4
 - <default line definition>, 4-4
 - <default line ID>, 4-4
- DEFAULT specification
 - for lines, 4-9
 - for stations, 4-21
 - for terminals, 4-32
- DEFAULT STATION definition, 4-4
 - <default station definition>, 4-4
 - <default station ID>, 4-4
- DEFAULT TERMINAL definition, 4-5
 - <default terminal definition>, 4-5
 - <default terminal ID>, 4-5
- DEFINE declaration, 2-8
 - <define declaration>, 2-8
 - <define expression token>, 2-8
 - <define ID>, 2-8
 - <define statement token>, 2-9
 - <define statement>, 2-8
 - <define text>, 2-8
- defining line control, 3-36
- DELETE option in compiler control, B-5
 - <delete option>, B-5
- DEQUEUE statement, 3-59
 - <dequeue statement>, 3-59
 - <destination CB variable>, 3-66
- DESTINATION editor variable, 3-24
 - <destination maxsize>, 3-25
- DIAGNOSE TRANSMITTER structure
 - variable, 3-85
 - <diagnostic rate>, 3-11
- DIAGNOSTICRATE line class attribute, 3-11
 - <dial digit>, 4-15
 - <dial sequence>, 4-15
 - <digit>, definition, 2-2
- DISABLE statement, 3-90
 - <disable statement>, 3-90
- DISCARDRESULT statement, 3-59

- <discardresult statement>, 3-59
- DISCONNECTACTION attribute, 4-11
- DISCONNECTACTION LINE structure
 - variable, 3-42
- DISCONNECTPOLICY enumerated type, 2-11
- DIV editor function, 3-27
 - <div function>, 3-27
 - <div modulus>, 3-27
- DLEDETECT RECEIVER structure variable, 3-81
- DO statement, 2-41
 - <do statement>, 2-41
- DPRDELAY attribute, 4-11
 - <drop count>, 3-27
- DROP editor function, 3-27
 - <drop function>, 3-27
- <duplicate structure declaration>, 3-35
- DUPLICATE structure definition, 3-35

E

- EBCDIC string, 2-5
 - <EBCDIC string>, definition, 2-5
- ECHOPLEX RECEIVER structure variable, 3-81
- editor
 - declarations, 3-23
 - definition, 1-7
 - description, 3-22
 - functions
 - DIV, 3-27
 - DROP, 3-27
 - HEAD, 3-28
 - INTEGER, 3-28
 - MOD, 3-28
 - STRING, 3-29
 - STRINGADD, 3-29
 - STRINGSUBTRACT, 3-30
 - SUBSTRING, 3-30
 - TAIL, 3-31
 - TAKE, 3-31
 - TRANSLATEIN, 3-32
 - TRANSLATEOUT, 3-32
 - TRIM, 3-32
 - predeclared structure variables, 3-23
 - predeclared variables, 3-24
 - process fault termination, 3-33
 - processes
 - definition, 3-1
 - special functions, 3-26

- special statements, 3-24
- statements
 - ALLOCATE, 3-24
 - TRANSFER, 3-25
 - TRANSLATE, 3-26
- <editor allocate statement>, 3-25
- EDITOR attribute, 4-24
- <editor body>, 3-23
- <editor declaration list>, 3-23
- <editor declaration>, 3-22
- <editor ID>, 3-23
- <editor input process>, 3-23
- <editor output process>, 3-23
- <editor structure name>, 3-23
- <editor transfer statement>, 3-25
- <editor translate function>, 3-32
- <editor translate statement>, 3-26
- elements allowed in a procedure, D-2
- ENABLE statement, 3-91
- <enable statement>, 3-91
- ENABLED attribute, 4-25
- ENABLED RECEIVER structure variable, 3-81
- ENABLED STATION structure variable, 3-45
- ENABLED TRANSMITTER structure variable, 3-85
- <ending sequence number>, B-6
- ENDTEXT adaptor control variable, 3-88
- ENQUEUE statement, 3-60
- <enqueue statement>, 3-60
- enumerated assignment, 2-37
- <enumerated assignment>, 2-37
- <enumerated attribute>, 4-2
- <enumerated constant list>, 2-10
- <enumerated constant>, 2-10
- enumerated expression, 2-26
- <enumerated expression>, 2-26
- <enumerated function>, 2-26
- <enumerated initial value>, 2-13
- <enumerated primary>, 2-26
- <enumerated type ID>, 2-10
- <enumerated type>, 2-10
- predeclared, 2-11
- enumerated types
 - CBCATEGORY, 2-11
 - CODE_EXT_TYPE, 2-11
 - DISCONNECTPOLICY, 2-11
 - INPUTPOLICY, 2-11
 - predeclared, D-2
 - RTERROR, 2-11
 - WAITRESULTTYPE, 2-11

- <enumerated value set>, 2-10
- enumerated variable declaration, 2-13
- <enumerated variable declaration part>, 2-13
- <enumerated variable declaration>, 2-13
- <enumerated variable ID>, 2-13
- <enumerated variable>, 2-13
- ENUMERATION declaration, 2-10
- <enumeration declaration>, 2-10
- <equality operator>, 2-22
- error messages, A-1
- ERROR RECEIVER structure variable, 3-82
- ERROR TRANSMITTER structure variable, 3-85
- ERRORLIMIT option in compiler control, B-5
- <errorlimit option>, B-6
- ERRORLIST option in compiler control, B-6
- <errorlist option>, B-6
- EVENT declaration, 3-52
- <event declaration>, 3-52
- <event ID>, 3-52
- <event variable>, 3-52
- example
 - configuration module, 4-42
 - NDLII source program, 1-4
- executive program, 1-7
- <expected character>, 3-95
- expression
 - Boolean, 2-21
 - byte, 2-25
 - definition, 2-21
 - enumerated, 2-26
 - integer, 2-27
 - string, 2-32
- <expression>, 2-21
- EXTEND RECEIVER structure variable, 3-83
- EXTEND TRANSMITTER structure variable, 3-86
- <external line body>, 3-37
- <extract byte from string>, 2-25

F

- fault messages, A-1
- fault termination
 - adaptor control, 3-107
 - editor process, 3-33
 - line control, 3-77
- FETCH statement, 3-92

<fetch statement>, 3-92
 FETCHPARITY adaptor control variable,
 3-88
 <field part>, 2-27
 <field start>, 2-27
 <field width>, 2-27
 <field>, 2-8
 <file attribute list>, 4-6
 FILE definition in configuration module, 4-6
 <file definition>, 4-6
 <file ID>, 4-6
 <files attribute>, 4-6
 <final polynomial>, 3-13
 FINISHREQUEST statement, 3-61
 <finishrequest statement>, 3-61
 FIRSTERROR RECEIVER structure
 variable, 3-83
 FIRSTERROR TRANSMITTER structure
 variable, 3-86
 <fixed variable declaration list>, 3-16
 <force DLE>, 3-99
 FUNCTION attribute, 4-12
 functions
 adaptor control, 3-104
 BCSOF, 3-104
 general integer, D-2
 HAPPENED, 3-74
 LENGTH, 2-28
 masking, 2-29
 MAX, 2-29
 MIN, 2-30
 NULLREFERENCE, 3-74
 ROTATELEFT, 2-30
 ROTATERIGHT, 2-30
 SHIFLEFT, 2-30
 SHIFTRIGHT, 2-30
 special
 integer, 2-31
 line control, 3-74
 TIMEINTERVAL, 3-75
 TRANSLATEIN
 in adaptor control, 3-105
 TRANSLATEOUT
 in adaptor control, 3-105
 WAIT, 3-76
 WAITFORIDLE, 3-106

G

<global reply declaration>, 3-16
 <global signal declaration>, 3-16

<group algorithm attribute>, 4-7
 <group attribute list>, 4-7
 group attributes in configuration module
 ALGORITHM, 4-7
 INITIALIZE, 4-8
 LINE ID, 4-8
 GROUP definition in configuration module,
 4-7
 <group definition>, 4-7
 <group ID>, 4-7
 <group initialize attribute>, 4-8
 <group line ID attribute>, 4-8
 <group part>, 3-15
 GROUP structure
 predeclared variables, 3-41
 LINECOUNT, 3-41
 MAXINPUT, 3-41

H

HAPPENED function, 3-74
 <happened function>, 3-74
 HARDWARE DEFINITION in configuration
 module, 4-39
 <hardware definition>, 4-39
 <hardware module>, 4-39
 HEAD
 editor function, 3-28
 <head function>, 3-28
 <hex character>, definition, 2-2
 <hex digit>, definition, 2-2
 hex string, 2-5
 <hex string>, definition, 2-5
 <host NSP ID>, 4-39

I

<I-field blocking>, 3-99
 <ID>, 2-8
 identifier, definition of, 2-1
 <identifier>, definition, 2-2
 IDLEDETECT RECEIVER structure
 variable, 3-83
 IF statement, 2-42
 <if statement>, 2-42
 <immediate option>, B-3
 INCLUDE declaration, 3-15
 in line control, 3-53
 <include declaration>, 3-15
 INCLUDE option in compiler control, B-6

<include option>, B-6
 indexes, used with translate tables, 3-19
 INHIBIT RECEIVER structure variable,
 3-83
 INHIBIT TRANSMITTER structure
 variable, 3-87
 <initial polynomial>, 3-13
 INITIALIZE
 attribute
 for groups, 4-8
 for lines, 4-13
 for stations, 4-25
 statement, 3-92
 <initialize statement>, 3-92
 INITIATE statement, 3-61
 <initiate statement>, 3-61
 INPUT process, 3-22
 <input process declaration>, 3-79
 INPUTACTION attribute, 4-26
 INPUTACTION STATION structure
 variable, 3-45
 INPUTPOLICY enumerated type, 2-11
 integer
 assignment, 2-38
 expression, 2-27
 functions
 general, D-2
 special, 2-31
 INTEGER
 declaration, 2-14
 editor function, 3-28
 <integer assignment>, 2-38
 <integer attribute>, 4-2
 <integer bit number>, 2-22
 integer constant, 2-5
 <integer constant>, definition, 2-5
 <integer declaration>, 2-14
 <integer expression>, 2-27
 <integer function>, 3-28
 <integer ID>, 2-14
 <integer initial value>, 2-14
 <integer primary>, 2-27
 <integer variable>, 2-14
 INTERLOCK declaration, 3-53
 <interlock declaration>, 3-53

K

<key>, 3-59
 <keyword>, definition, 2-3

L

language
 components
 common, 2-1
 NDLII, D-1
 elements, definitions, 2-2
 LENGTH function, 2-28
 <length function>, 2-28
 <letter>, definition, 2-3
 LINE
 definition in configuration module, 4-9
 process, 3-36
 <line algorithm attribute>, 4-10
 <line attribute declaration>, 4-2
 <line attribute list>, 4-9
 line attributes in configuration module
 ALGORITHM, 4-10
 CLASS, 4-10
 DISCONNECTACTION, 4-11
 DPRDELAY, 4-11
 FUNCTION, 4-12
 INITIALIZE, 4-13
 LOSSOFCARRIER, 4-14
 PHONE, 4-15
 READY, 4-16
 RECEIVEDDELAY, 4-16
 SETDIGITDELAY, 4-17
 STATION list, 4-18
 TIMEOUT, 4-19
 TRANSMITDELAY, 4-19
 <line body>, 3-37
 line class, 3-4
 <line class attribute>, 4-10
 line class attributes
 ADDRESSMODE, 3-9
 BITRATE, 3-10
 CODE, 3-10
 CONTROLMODE, 3-11
 descriptions, 3-9
 DIAGNOSTICRATE, 3-11
 NRZI, 3-11
 PARITY, 3-12
 STOPBITS, 3-14
 SYNCCONTROL, 3-14
 <line class>, 3-4
 line control
 declarations, 3-38
 special, 3-51
 description, 3-36
 fault termination, 3-77
 INCLUDE declarations, 3-53

- predeclared variables, 3-49
 - NOMEMORY, 3-49
 - SYSTEMWAIT, 3-49
 - TIMESTAMP, 3-49
- PROCESS declarations, 3-54
- processes, definition, 3-1
- responsibilities, 1-8
- special functions, 3-74
- special statements, 3-55
- STRUCTURE declarations, 3-54
- variable declarations, 3-54
- LINE CONTROL definition, 3-36
 - < line control ID >, 3-36
 - < line control structure name >, 3-38
 - < line control >, 3-36
 - < line declaration list >, 3-37
- line DEFAULT specification in configuration
 - module, 4-9
- < line default specification >, 4-9
- < line definition >, 4-9
- < line disconnectaction attribute >, 4-11
- < line dprdelay attribute >, 4-11
- < line function attribute >, 4-12
- LINE ID attribute, 4-8
 - < line ID >, 4-9
- < line initialize attribute >, 4-13
- < line lossofcarrier attribute >, 4-14
- < line part >, 3-15
- < line phone attribute >, 4-15
- < line process >, 3-37
- < line ready attribute >, 4-16
- < line receivedelay attribute >, 4-16
- < line setdigitdelay attribute >, 4-17
- < line station list >, 4-18
- LINE structure
 - predeclared variables, 3-42
 - ADAPTOREVENT, 3-42
 - ADAPTORREADY, 3-42
 - BUSY, 3-42
 - CONNECTED, 3-42
 - DISCONNECTACTION, 3-42
 - READY, 3-43
 - STATUSCHANGED, 3-43
 - SYSTEM, 3-44
 - < line timeout attribute >, 4-19
 - < line transmitdelay attribute >, 4-19
- LINECHAR
 - RECEIVER structure variable, 3-84
 - TRANSMITTER structure variable, 3-87
- LINECOUNT GROUP structure variable, 3-41
- LINEINFO option in compiler control, B-7

- < lineinfo option >, B-7
- LINEWIDTH attribute, 4-34
- LINEWIDTH STATION structure variable, 3-45
- < list option >, B-8
- LIST\$ option in compiler control, B-8
- < list\$ option >, B-8
- LISTDELETED option in compiler control, B-8
- < listdeleted option >, B-8
- LISTINCL option in compiler control, B-8
- < listincl option >, B-8
- LISTP option in compiler control, B-8
- < listp option >, B-9
- < literal >, definition, 2-3
- LOADSTRUCTURE statement, 3-93
 - < loadstructure statement >, 3-93
- < lock ID >, 3-53
- LOCK statement, 3-64
 - < lock statement >, 3-64
- < lock variable >, 3-53
- locks, declaring, 3-53
- logical operator, 2-23
 - precedence, 2-24
- < logical operator >, 2-22
 - precedence, 2-24
 - truth table, 2-23
- LOSSOFCARRIER attribute, 4-14
- < LSP adaptor declaration >, 4-39
- < LSP adaptor number >, 4-39
- < LSP alternates >, 4-39
- < LSP ID >, 4-39
- < LSP module >, 4-39

M

- MAP option in compiler control, B-9
 - < map option >, B-9
 - < map value >, 2-10
- masking function, 2-29
 - < masking function >, 2-29
 - truth table, 2-29
- MAX function, 2-29
 - < max function >, 2-29
 - < max size >, 4-2
- MAXINPUT attribute, 4-34
- MAXINPUT GROUP structure variable, 3-41
- MAXINPUT STATION structure variable, 3-45
- MAXOUTPUT attribute, 4-35

MAXOUTPUT STATION structure variable,
 3-45
MCS, (See message control system (MCS))
MERGE option in compiler control, B-9
 <merge option>, B-9
message control system (MCS), 1-10
messages
 request, 1-7
 result, 1-7
MIN function, 2-30
 <min function>, 2-30
MOD editor function, 3-28
 <mod function>, 3-29
 <mod modulus>, 3-29
mode
 asynchronous, 3-5
 bit, 3-6
 synchronous, 3-7
MOVETEXT statement, 3-65
 <movetext statement>, 3-66
MYUSE attribute, 4-26

N

NDLII, (See Network Definition Language II
 (NDLII))
 <NDLII analyzer options>, E-2
NDLII compiler, (See compiler)
NDLII post compiler, (See post compiler)
NDLIIANALYZER, E-1
 display formats, E-5
 error messages, E-3
 files, E-1
 initiation, E-1
 options, E-2
Network Definition Language II (NDLII), 1-2
 source program, 1-2
 example of, 1-4
network information file II (NIFII), 1-6
NEW option in compiler control, B-10
 <new option>, B-10
NEWRESULT statement, 3-67
 <newresult statement>, 3-67
NEXTREQUEST statement, 3-67
 <nextrequest statement>, 3-67
NEXTSTATION statement, 3-68
 <nextstation statement>, 3-68
NIFII, (See network information file II
 (NIFII))
NOCANCEL adaptor control variable, 3-88
NOMEMORY line control variable, 3-49

NRZI line class attribute, 3-11
 <NRZI>, 3-11
 <NSP ID>, 4-39
null string, 2-5
 <null string>, definition, 2-5
NULLREFERENCE function, 3-74
 <nullreference function>, 3-74
NULLSTATION statement, 3-69
 <nullstation statement>, 3-69
numeric constant, 2-5
 <numeric constant>, definition, 2-5

O

 <octal digit>, definition, 2-3
 <operation type>, 3-61
operators, valid Boolean, D-2
 <option list>, 4-39
OUTPUT process, 3-22
 <output process declaration>, 3-79
OUTPUTLIMIT attribute, 4-27

P

PAGE option in compiler control, B-10
 <page option>, B-10
PAGECOUNT attribute, 4-35
PAGECOUNT STATION structure variable,
 3-46
PAGESIZE attribute, 4-36
PAGESIZE option in compiler control, B-10
 <pagesize option>, B-11
PAGESIZE STATION structure variable,
 3-46
parity checking, 3-12
PARITY line class attribute, 3-12
 <parity>, 3-13
 <patch number>, B-14
PHONE attribute
 for lines, 4-15
 for stationsets, 4-31
 <polynomial declaration>, 3-13
 <polynomial expression>, 3-13
 <polynomial term>, 3-13
 <polynomial>, 3-13
post compiler, B-1
 card file user options, B-21
 configuration overview, B-18
 description, B-16
 files, B-17

- limits, B-16
- running, B-18
- using CANDE with, B-19
- precedence
 - complement operator, 2-24
 - conditional operator, 2-24
 - logical operator, 2-24
 - of operators, 2-24
- predeclared enumerated types, D-2
- predeclared variables
 - adaptor control, 3-88
 - BUFFEROVERFLOW, 3-88
 - ENDTEXT, 3-88
 - FETCHPARITY, 3-88
 - NOCANCEL, 3-88
 - control block (CB) structure, 3-39, 3-81
 - GROUP structure, 3-41
 - GROUP structure variables
 - LINECOUNT, 3-41
 - MAXINPUT, 3-41
 - line control, 3-49
 - NOMEMORY, 3-49
 - SYSTEMWAIT, 3-49
 - TIMESTAMP, 3-49
 - LINE structure, 3-42
 - ADAPTOREVENT, 3-42
 - ADAPTORREADY, 3-42
 - BUSY, 3-42
 - CONNECTED, 3-42
 - DISCONNECTACTION, 3-42
 - READY, 3-43
 - STATUSCHANGED, 3-43
 - SYSTEM, 3-44
 - RECEIVER structure, 3-81
 - BCS, 3-81
 - DLEDETECT, 3-81
 - ECHOPLEX, 3-81
 - ENABLED, 3-81
 - ERROR, 3-82
 - EXTEND, 3-83
 - FIRSTERROR, 3-83
 - IDLEDETECT, 3-83
 - INHIBIT, 3-83
 - LINECHAR, 3-84
 - RESIDUE, 3-84
 - SPECIAL, 3-84
 - TRANSLATE, 3-84
 - TRANSPARENT, 3-84
 - STATION structure, 3-45
 - CONTROL, 3-45
 - ENABLED, 3-45
 - INPUTACTION, 3-45
 - LINEWIDTH, 3-45
 - MAXINPUT, 3-45
 - MAXOUTPUT, 3-45
 - PAGECOUNT, 3-46
 - PAGESIZE, 3-46
 - QUEUED, 3-46
 - READY, 3-46
 - RECEIVEADDRESS, 3-46
 - REQUEST, 3-46
 - REQUESTLIST, 3-47
 - RESULT, 3-47
 - RESULTLIST, 3-48
 - SCREEN, 3-48
 - TRANSMITADDRESS, 3-48
 - TYPE, 3-48
 - USAGECOUNT, 3-48
 - WRAPAROUND, 3-48
 - TRANSMITTER structure, 3-85
 - BCS, 3-85
 - DIAGNOSE, 3-85
 - ENABLED, 3-85
 - ERROR, 3-85
 - EXTEND, 3-86
 - FIRSTERROR, 3-86
 - INHIBIT, 3-87
 - LINECHAR, 3-87
 - RESIDUE, 3-87
 - SPECIAL, 3-87
 - TRANSLATE, 3-87
 - TRANSPARENT, 3-87
 - <primary>, 2-8
 - <procedure body>, 2-15
 - PROCEDURE declaration, 2-15
 - <procedure declaration list>, 2-15
 - <procedure declaration>, 2-15
 - <procedure ID>, 2-15
 - PROCESS
 - declaration, 2-16
 - statement, 3-69
 - <process body>, 2-16
 - <process declaration list>, 2-16
 - <process declaration>, 2-16
 - <process ID>, 2-16
 - <process statement>, 3-69
 - processes
 - declarations, adaptor control, 3-88
 - definition, 2-16
 - editor, definition, 3-1
 - INPUT, 3-22
 - line control, definition, 3-1
 - OUTPUT, 3-22
 - required, 2-16

protocol module, 3-1
 declarations, 3-2
 recognized words, list of, C-2
 reserved words, list of, C-1
 structure, 1-3
<protocol module>, 3-2
<purge selector>, 3-70
PURGE statement, 3-70
<purge statement>, 3-70

Q

qualified variables, 2-20
<qualifier>, 2-20
QUEUED STATION structure variable, 3-46

R

railroad diagrams, explanation of, F-1
READY attribute
 for lines, 4-16
 for stations, 4-27
READY LINE structure variable, 3-43
READY STATION structure variable, 3-46
REBLOCK statement, 3-71
<reblock statement>, 3-71
<receive address field>, 3-94
<receive address size>, 4-33
<receive control field>, 3-94
<receive frame field part>, 3-94
<receive I-field>, 3-94
<receive item>, 3-94
RECEIVE statement, 3-94
<receive statement>, 3-94
<receive terminate list>, 3-94
<receive timeout>, 3-94
RECEIVEADDRESS STATION structure variable, 3-46
RECEIVEDELAY attribute
 for lines, 4-16
 for terminals, 4-36
RECEIVER structure
 predeclared variables, 3-81
 BCS, 3-81
 DLEDETECT, 3-81
 ECHOPLEX, 3-81
 ENABLED, 3-81
 ERROR, 3-82
 EXTEND, 3-83

FIRSTERROR, 3-83
IDLEDETECT, 3-83
INHIBIT, 3-83
LINECHAR, 3-84
RESIDUE, 3-84
SPECIAL, 3-84
TRANSLATE, 3-84
TRANSPARENT, 3-84
receiver, operation, 3-4
 <recognized word>, 2-3
recognized words, list of, C-1
<reference variable>, 3-74
<relational operator>, 2-22
<release number>, B-14
<repeat count>, 3-25
<repeat part>, 3-25
REPLY declaration, 3-16, 3-35
 <reply declaration>, 3-35
REPLY INPUT area, 3-16
REPLY OUTPUT area, 3-16
request messages, 1-7
REQUEST STATION structure variable, 3-46
 <request-to-CB>, 3-66
REQUESTLIST STATION structure variable, 3-47
 <reserved word>, definition, 2-3
reserved words, list of, C-1
 <residue in>, 3-71
 <residue out>, 3-71
RESIDUE RECEIVER structure variable, 3-84
RESIDUE TRANSMITTER structure variable, 3-87
 <result length>, 3-29
result messages, 1-7
RESULT STATION structure variable, 3-47
 <result type>, 3-72
RESULTLIST STATION structure variable, 3-48
 <rotate count>, 2-30
 <rotate function>, 2-30
ROTATELEFT function, 2-30
ROTATERIGHT function, 2-30
RTERROR enumerated type, 2-11

S

SCREEN attribute, 4-37
SCREEN STATION structure variable, 3-48
 <select address>, 4-12

- <select part>, 3-68
- <selected declaration list>, 3-17
- SENDBOST statement, 3-72
- <sendhost statement>, 3-72
- <seqcheck option>, B-11
- <sequence base option>, B-12
- <sequence increment option>, B-12
- SEQUENCE option in compiler control, B-11
- <sequence option>, B-11
- SETDIGITDELAY attribute, 4-17
- <shift count>, 2-31
- <shift function>, 2-31
- SHIFLEFT function, 2-30
- SHIFTRIGHT function, 2-30
- SHORTEN statement, 2-43
- <shorten statement>, 2-43
- SIGNAL declaration, 3-16, 3-35
- <signal declaration>, 3-35
- SIGNAL OUTPUT area, 3-16
- <signal-reply Boolean declaration>, 2-6
- <signal-reply byte declaration>, 2-7
- <signal-reply declaration list>, 3-17
- <signal-reply enumerated declaration>, 2-13
- <signal-reply integer declaration>, 2-14
- <signal-reply string declaration>, 2-17
- SKIP statement, 2-43
- <skip statement>, 2-43
- <source CB variable>, 3-66
- SOURCE editor variable, 3-24
- <source structure>, 3-58
- SOURCENDLII, 1-5
 - submitting comments or suggestions, vi
- special adaptor control functions, 3-104
- <special adaptor statement>, 3-89
- <special Boolean function>, 2-22
- <special byte function>, 2-25
- <special editor statement>, 3-24
- special functions, line control, 3-74
- <special integer function>, 2-31
- <special line statement>, 3-55
- <special operation>, 3-61
- SPECIAL RECEIVER structure variable, 3-84
 - <special string function>, 2-33
 - <special string primary>, 2-33
 - <special token>, definition, 2-3
- SPECIAL TRANSMITTER structure variable, 3-87
- <starting sequence number>, B-6
- statements, 2-35
 - ABORT, 2-35
 - adaptor control
 - list, D-7
 - special, 3-89
 - ALLOCATE, 3-56
 - assignment, 2-36
 - CALL, 2-38
 - CANCEL, 3-89
 - CASE, 2-39
 - CAUSE, 3-56
 - CLEAR STATION, 3-57
 - compound, 2-40
 - COPYSTRING, 3-57
 - COPYSTRUCTURE, 3-58
 - DEALLOCATE, 3-58
 - DEQUEUE, 3-59
 - DISABLE, 3-90
 - DISCARDRESULT, 3-59
 - DO, 2-41
 - editor, D-4
 - ENABLE, 3-91
 - ENQUEUE, 3-60
 - FETCH, 3-92
 - FINISHREQUEST, 3-61
 - general, 2-35
 - IF, 2-42
 - INITIALIZE, 3-92
 - INITIATE, 3-61
 - line control, list of, D-5
 - LOADSTRUCTURE, 3-93
 - LOCK, 3-64
 - MOVETEXT, 3-65
 - NEWRESULT, 3-67
 - NEXTREQUEST, 3-67
 - NEXTSTATION, 3-68
 - NULLSTATION, 3-69
 - PROCESS, 3-69
 - PURGE, 3-70
 - REBLOCK, 3-71
 - RECEIVE, 3-94
 - SENDBOST, 3-72
 - SHORTEN, 2-43
 - simple, 2-35
 - SKIP, 2-43
 - special, 2-35
 - STORE, 3-98
 - structured, 2-35
 - TRACE, 3-72
 - TRANSMIT, 3-99
 - UNLOCK, 3-73
 - WAIT, 3-73
 - in adaptor control, 3-103
 - WHILE, 2-44

station

reference variables, 3-50

STATION

definition in configuration module, 4-20

< station address attribute >, 4-21

< station algorithm attribute >, 4-22

< station applicationlist attribute >, 4-23

< station attribute declaration >, 4-2

< station attribute list >, 4-20

station attributes in configuration module

ADDRESS, 4-21

ALGORITHM, 4-22

APPLICATIONLIST, 4-23

CONTROL, 4-24

EDITOR, 4-24

ENABLED, 4-25

INITIALIZE, 4-25

INPUTACTION, 4-26

MYUSE, 4-26

OUTPUTLIMIT, 4-27

READY, 4-27

TERMINAL, 4-28

TITLE, 4-28

< station control attribute >, 4-24

station DEFAULT specification in
configuration module, 4-21

< station default specification >, 4-21

< station definition >, 4-20

< station editor attribute >, 4-24

< station enabled attribute >, 4-25

< station ID >, 4-20

< station initialize attribute >, 4-25

< station inputaction attribute >, 4-26

STATION list, 4-18

< station myuse attribute >, 4-26

< station outputlimit attribute >, 4-27

< station part >, 3-15

< station ready attribute >, 4-27

STATION structure

predeclared variables, 3-45

CONTROL, 3-45

ENABLED, 3-45

INPUTACTION, 3-45

LINEWIDTH, 3-45

MAXINPUT, 3-45

MAXOUTPUT, 3-45

PAGECOUNT, 3-46

PAGESIZE, 3-46

QUEUED, 3-46

READY, 3-46

RECEIVEADDRESS, 3-46

REQUEST, 3-46

REQUESTLIST, 3-47

RESULT, 3-47

RESULTLIST, 3-48

SCREEN, 3-48

TRANSMITADDRESS, 3-48

TYPE, 3-48

USAGECOUNT, 3-48

WRAPAROUND, 3-48

< station terminal attribute >, 4-28

< station title attribute >, 4-28

STATIONS attribute, (See STATION list)

< stations attribute >, 4-6

stations, maximum number, 4-20

< stationset attribute >, 4-29

stationset attributes in configuration module

member list, 4-30

PHONE, 4-31

STATIONSET definition in configuration

module, 4-29

< stationset definition >, 4-29

< stationset ID >, 4-29

stationset member list, 4-30

< stationset member list >, 4-30

< stationset phone attribute >, 4-31

stationset STATIONS attribute, (See
stationset member list)

STATUSCHANGED LINE structure

variable, 3-43

stop bits, 3-14

< stop bits >, 3-14

STOPBITS line class attribute, 3-12, 3-14

STORE statement, 3-98

< store statement >, 3-98

string

assignment, 2-38

expression, 2-32

STRING

declaration, 2-17

editor functions, 3-29

< string assignment >, 2-38

< string attribute >, 4-2

string constant, 2-5

< string constant >, definition, 2-5

< string declaration part >, 2-17

< string declaration >, 2-17

< string expression >, 2-32

< string function >, 3-29

< string ID >, 2-17

< string initial value >, 2-17

< string max size >, 2-17

< string primary >, 2-32

< string subscript >, 2-25

<string variable>, 2-17
STRINGADD
 editor function, 3-29
 <stringadd function>, 3-30
STRINGSUBTRACT
 editor function, 3-30
 <stringsubtract function>, 3-30
structure
 definition, 2-18
 predeclared, 2-18
 <structure array ID>, 2-18
STRUCTURE declaration, 2-18
 <structure declaration part>, 3-15
 <structure declaration>, 2-18
 <structure element>, 2-18
 <structure ID>, 2-18
 <structure name>, 2-20
 <structure part>, 2-18
 <structure size>, 2-18
 <structure subscript>, 2-18
 <structure variable>, 2-18
 <structure>, 3-93
SUBSTRING
 editor functions, 3-30
 <substring function>, 3-31
 <substring length>, 3-31
 <substring start>, 3-31
 <subtract result length>, 3-30
SUMMARY option in compiler control, B-12
 <summary option>, B-13
SUPPRESSACTION editor structure
 variable, 3-23
 <sync class>, 3-8
 <sync control>, 3-14
SYNCCONTROL line class attribute, 3-14
synchronous mode, 3-7
 operation, 3-8
SYSTEM LINE structure variable, 3-44
system messages, A-1
SYSTEMWAIT line control variable, 3-49

T

TAIL
 editor function, 3-31
 <tail function>, 3-31
TAKE
 editor function, 3-31
 <take count>, 3-32
 <take function>, 3-32
 <template guard>, 3-17

 <template selection variable>, 3-17
 <template selection>, 3-17
 <template>, 3-16
TERMINAL
 attribute, 4-28
 definition in configuration module, 4-32
 <terminal addresslength attribute>, 4-33
 <terminal attribute declaration>, 4-2
 <terminal attribute list>, 4-32
 terminal attributes in configuration module
 ADDRESSLENGTH, 4-33
 CLASS, 4-33
 LINEWIDTH, 4-34
 MAXINPUT, 4-34
 MAXOUTPUT, 4-35
 PAGECOUNT, 4-35
 PAGESIZE, 4-36
 RECEIVEDDELAY, 4-36
 SCREEN, 4-37
 TRANSMITDELAY, 4-37
 TYPE, 4-38
 WRAPAROUND, 4-38
 <terminal class attribute>, 4-33
 terminal **DEFAULT** specification in
 configuration module, 4-32
 <terminal default specification>, 4-32
 <terminal definition>, 4-32
 <terminal ID>, 4-32
 <terminal linewidth attribute>, 4-34
 <terminal maxinput attribute>, 4-34
 <terminal maxoutput attribute>, 4-35
 <terminal pagecount attribute>, 4-35
 <terminal pagesize attribute>, 4-36
 <terminal receivedelay attribute>, 4-36
 <terminal screen attribute>, 4-37
 <terminal transmitdelay attribute>, 4-37
 <terminal type attribute>, 4-38
 <terminal wraparound attribute>, 4-38
TERMINALTYPE declaration, 3-18
TEXT
 control block (CB) structure variable, 3-81
 control block (CB) variable, 3-40
TEXTSIZE control block variable, 3-40
 <time part>, 3-94
TIMEINTERVAL function, 3-75
 <timeinterval function>, 3-75
TIMEOUT attribute, 4-19
 <timeout>, 3-106
TIMESTAMP line control variable, 3-49
 <timestamp string>, 3-75
TITLE
 attribute, 4-28

- option in compiler control, B-13
- <title attribute>, 4-6
- <title option>, B-13
- <title string>, B-13
- <token>, definition, 2-3
- TRACE statement, 3-72
- <trace statement>, 3-72
- TRAINID option in compiler control, B-13
- <trainid option>, B-13
- TRANSFER
 - editor statement, 3-25
- TRANSLATE
 - editor statement, 3-26
 - RECEIVER structure variable, 3-84
 - TRANSMITTER structure variable, 3-87
- <translate body>, 3-19
- <translate clause>, 3-19
- translate tables, 3-19
 - indexes, 3-19
- TRANSLATEIN
 - editor function, 3-32
 - function, in adaptor control, 3-105
- TRANSLATEOUT
 - editor function, 3-32
 - function, in adaptor control, 3-105
- TRANSLATETABLE declaration, 3-19
- <translatetable declaration>, 3-19
- <translatetable ID>, 3-19
- <translatetable>, 3-19
- translation, 3-19
 - <transmit address field>, 3-99
 - <transmit address size>, 4-33
 - <transmit control field>, 3-99
 - <transmit frame field part>, 3-99
 - <transmit I-field>, 3-99
 - <transmit item>, 3-99
- TRANSMIT statement, 3-99
- <transmit statement>, 3-99
- <transmit termination list>, 3-99
- TRANSMITADDRESS STATION structure
 - variable, 3-48
- TRANSMITDELAY attribute
 - line, 4-19
 - terminal, 4-37
- TRANSMITTER structure
 - predeclared variables, 3-85
 - BCS, 3-85
 - DIAGNOSE, 3-85
 - ENABLED, 3-85
 - ERROR, 3-85
 - EXTEND, 3-86
 - FIRSTERROR, 3-86

- INHIBIT, 3-87
- LINECHAR, 3-87
- RESIDUE, 3-87
- SPECIAL, 3-87
- TRANSLATE, 3-87
- TRANSPARENT, 3-87
- transmitter, operation, 3-4
- TRANSPARENT RECEIVER structure
 - variable, 3-84
- TRANSPARENT TRANSMITTER structure
 - variable, 3-87
- TRIM editor function, 3-32
- <trim function>, 3-32
- truth table
 - complement operator, 2-23
 - conditional operator, 2-23
 - logical operator, 2-23
 - masking function, 2-29
- TYPE attribute, 4-38
- TYPE STATION structure variable, 3-48

U

- UNLOCK statement, 3-73
- <unlock statement>, 3-73
- UPCASELISTING option in compiler control,
 - B-14
- <upcaselisting option>, B-14
- USAGECOUNT STATION structure
 - variable, 3-48

V

- <value option>, B-3
- values
 - bit rate, 3-10
- variable declaration list, 2-19
- <variable declaration list>, 2-19
- variable declarations
 - in line control, 3-54
- <variable>, 2-9
- variables
 - <binary digit>, 2-2
 - <Boolean constant>, 2-4
 - <byte constant>, 2-4
 - <byte mnemonic>, 2-4
 - <character>, 2-2
 - <constant>, 2-2
 - <digit>, 2-2
 - <EBCDIC string>, 2-5

<hex character>, 2-2
 <hex digit>, 2-2
 <hex string>, 2-5
 <identifier>, 2-2
 <integer constant>, 2-5
 <keyword>, 2-3
 <letter>, 2-3
 <literal>, 2-3
 <null string>, 2-5
 <numeric constant>, 2-5
 <octal digit>, 2-3
 qualified, 2-20
 <recognized word>, 2-3
 <reserved word>, 2-3
 <special token>, 2-3
 station reference, 3-50
 <string constant>, 2-5
 <token>, 2-3
 <version number>, B-14
 VERSION option in compiler control, B-14
 <version option>, B-14
 VOID option in compiler control, B-15
 <void option>, B-15
 <xref option>, B-16

W

<wait event>, 3-73
 WAIT function, 3-76
 <wait function>, 3-76
 <wait parameter list>, 3-76
 WAIT statement, 3-73
 in adaptor control, 3-103
 <wait statement>, 3-73
 <wait timeout>, 3-76
 WAITFORIDLE function, 3-106
 <waitforidle function>, 3-106
 WAITRESULTTYPE enumerated type, 2-11
 WARNFATAL option in compiler control,
 B-15
 <warnfatal option>, B-15
 WHILE statement, 2-44
 <while statement>, 2-44
 WRAPAROUND attribute, 4-38
 WRAPAROUND STATION structure
 variable, 3-48

X

XREF option in compiler control, B-16



116960400P005



1169604000000380